EXHIBIT 1

United States Patent [19]

Kobayashi et al.

[11] **Patent Number: 4,922,432**

[45] **Date of Patent: May 1, 1990**

[54] **KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS**

[75] Inventors: **Hideaki Kobayashi**, Columbia, S.C.; **Masahiro Shindo**, Osaka, Japan

[73] Assignees: **International Chip Corporation**, Columbia, S.C.; **Ricoh Company, Ltd.**, Tokyo, Japan

[21] Appl. No.: **143,821**

[22] Filed: **Jan. 13, 1988**

[51] **Int. Cl.**[5] **G06F 15/60**

[52] **U.S. Cl.** **364/490**; 364/489; 364/488; 364/521

[58] **Field of Search** 364/488–491, 364/521, 300, 513

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,635,208	1/1987	Coleby et al.	364/491
4,638,442	1/1987	Bryant et al.	364/489
4,648,044	3/1987	Hardy et al.	364/513
4,651,284	3/1987	Watanabe et al.	364/491
4,656,603	4/1987	Dunn	364/488
4,658,370	4/1987	Erman et al.	364/513
4,675,829	6/1987	Clemenson	364/513
4,700,317	10/1987	Watanabe et al.	364/521
4,703,435	10/1987	Darringer et al.	364/488
4,803,636	2/1989	Nishiyama et al.	364/491

FOREIGN PATENT DOCUMENTS

1445914	8/1976	United Kingdom	364/490

OTHER PUBLICATIONS

"Verifying Compiled Silicon", by E. K. cheng, VLSI Design, Oct. 1984, pp. 1–4.

"CAD System for IC Design", by M. E. Daniel et al., IEEE Trans. on Computer-Aided Design of Integrated Circuits & Systems, vol. CAD–1, No. 1, Jan. 1982, pp. 2–12.

"An Overview of Logic Synthesis System", by L. Trevillyan, 24th ACM/IEEE Design Automation Conference, 1978, pp. 166–172.

"Methods Used in an Automatic Logic Design Generator", by T. D. Friedman et al., IEEE Trans. on Computers, vol. C–18, No. 7, Jul. 1969, pp. 593–613.

"Experiments in Logic Synthesis", by J. A. Darringer, IEEE ICCC, 1980.

"A Front End Graphic Interface to First Silicon Compiler", by J. H. Nash, EDA 84, Mar. 1984.

"quality of Designs from An Automatic Logic Generator", by T. D. Friedman et al., IEEE 7th DA Conference, 1970, pp. 71–89.

"A New Look at Logic Synthesis", by J. A. Darringer et al., IEEE 17th D. A. Conference 1980, pp. 543–548.

Trevillyan—Trickey, H., Flamel: *A High Level Hardward Compiler*, IEEE Transactions On Computer Aided Design, Mar. 1987, pp. 259–269.

Parker et al., The *CMU Design Automation System—An Example of Automated Data Path Design*, Proceedings Of The 16th Design Automation Conference, Las Vegas, Nev., 1979, pp. 73–80.

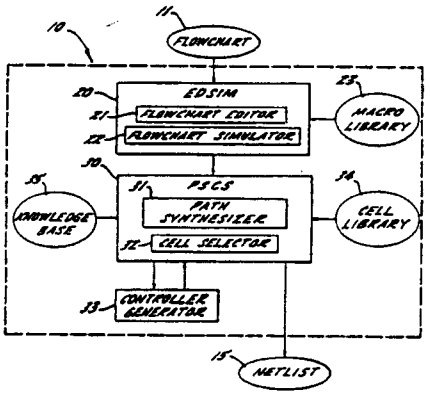
An Engineering Approach to Digital Design, William I. Fletcher, Prentice-Hall, Inc., pp. 491–505.

Primary Examiner—Felix D. Gruber
Assistant Examiner—V. N. Trans
Attorney, Agent, or Firm—Bell, Seltzer, Park & Gibson

[57] **ABSTRACT**

The present invention provides a computer-aided design system and method for designing an application specific integrated circuit which enables a user to define functional architecture independent specifications for the integrated circuit and which translates the functional architecture independent specifications into the detailed information needed for directly producing the integrated circuit. The functional architecture independent specifications of the desired integrated circuit can be defined at the functional architecture independent level in a flowchart format. From the flowchart, the system and method uses artificial intelligence and expert systems technology to generate a system controller, to select the necessary integrated circuit hardware cells needed to achieve the functional specifications, and to generate data and control paths for operation of the integrated circuit. This list of hardware cells and their interconnection requirements is set forth in a netlist. From the netlist it is possible using known manual techniques or existing VLSI CAD layout systems to generate the detailed chip level topological information (mask data) required to produce the particular application specific integrated circuit.

20 Claims, 12 Drawing Sheets



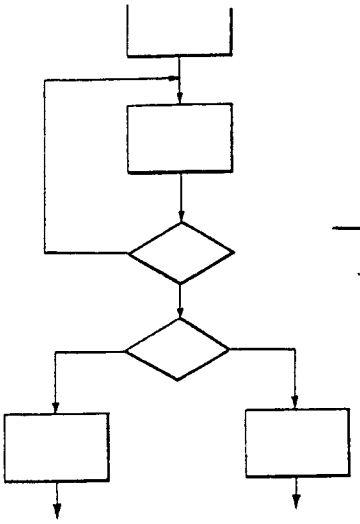


Fig. 1a.
FUNCTIONAL LEVEL

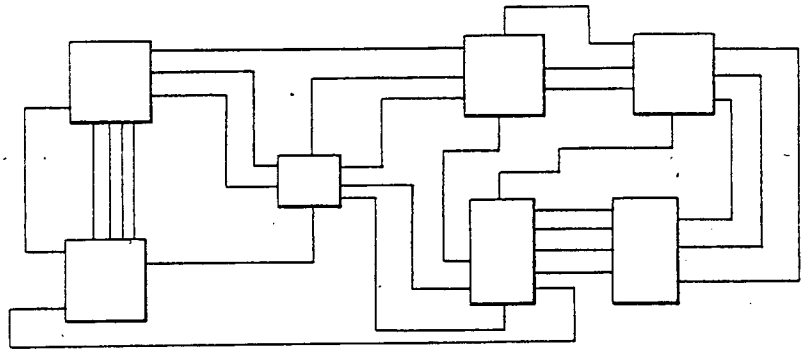


Fig. 1b.
STRUCTURAL LEVEL

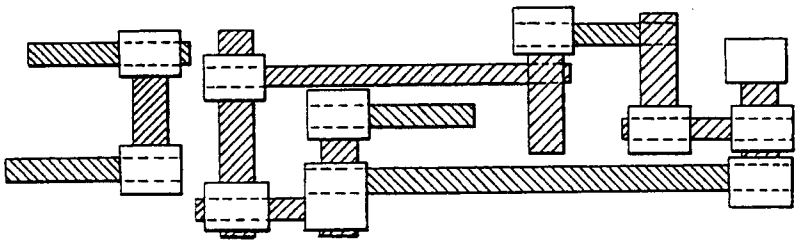


Fig. 1c.
PHYSICAL LAYOUT LEVEL

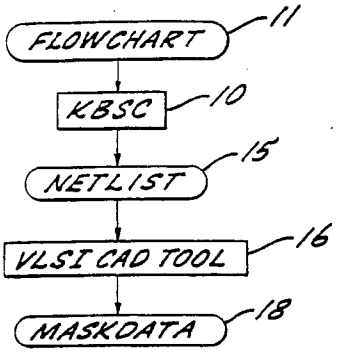

FLOWCHART
11
KBSC
10
NETLIST
15
VLSI CAD TOOL
16
MASKDATA
18

Fig. 2.

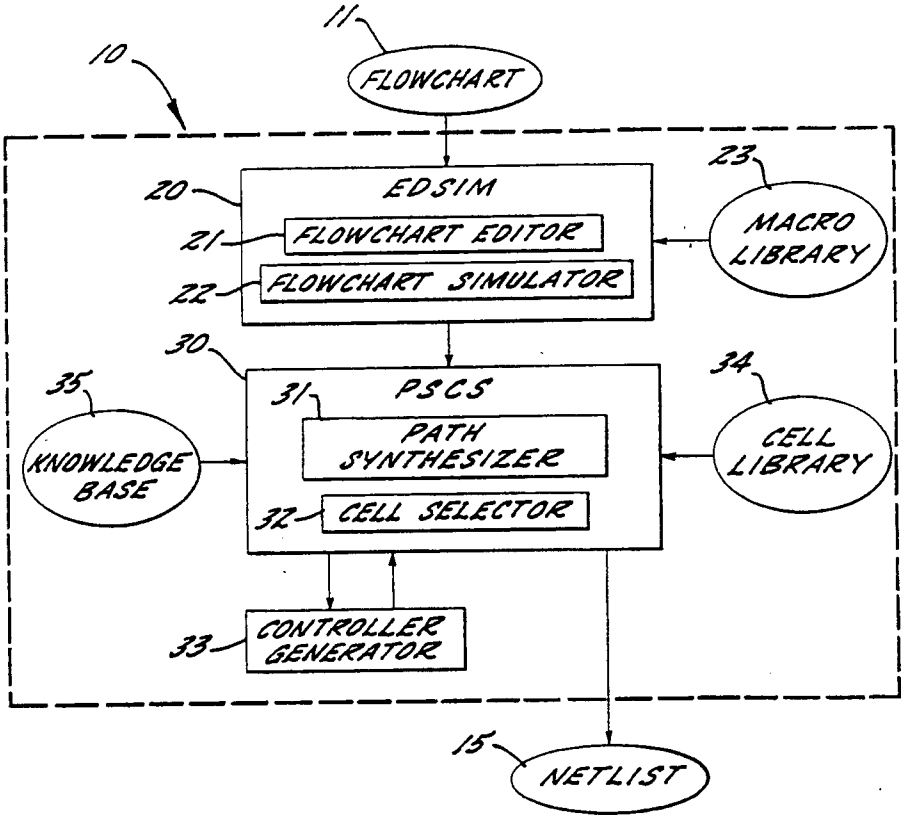

11
FLOWCHART
10
EDSIM
20
FLOWCHART EDITOR
21
FLOWCHART SIMULATOR
22
23
MACRO LIBRARY
30
PSCS
31
PATH SYNTHESIZER
32
CELL SELECTOR
33
CONTROLLER GENERATOR
35
KNOWLEDGE BASE
34
CELL LIBRARY
15
NETLIST

Fig. 3.

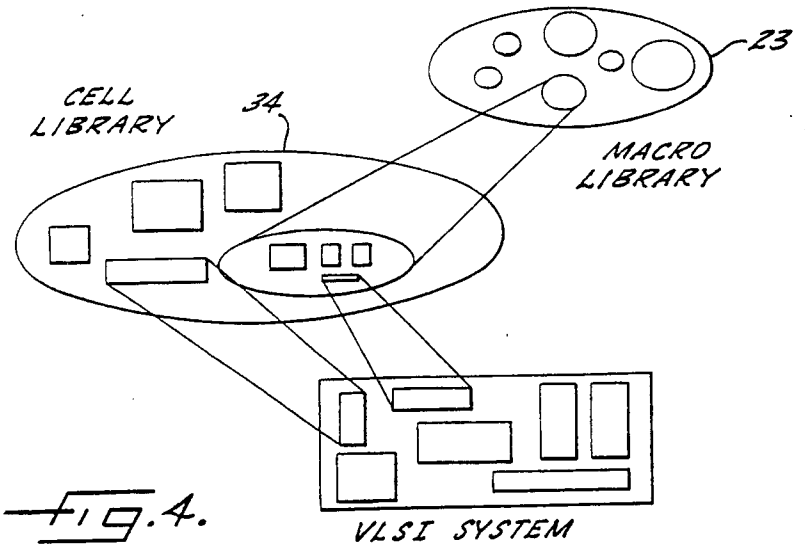

23
CELL LIBRARY
34
MACRO LIBRARY
VLSI SYSTEM

Fig. 4.

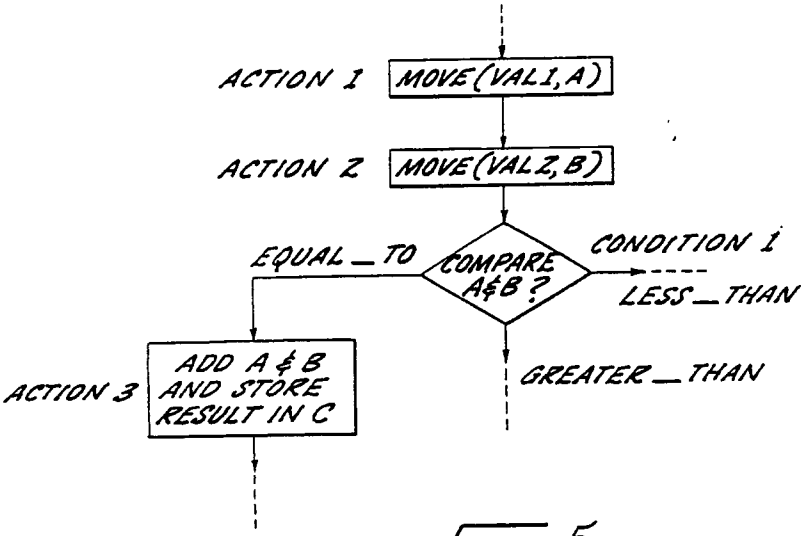

ACTION 1
MOVE(VAL1,A)
ACTION 2
MOVE(VAL2,B)
EQUAL_TO
COMPARE A$B ?
CONDITION 1
LESS_THAN
GREATER_THAN
ACTION 3
ADD A $ B AND STORE RESULT IN C

Fig. 5.

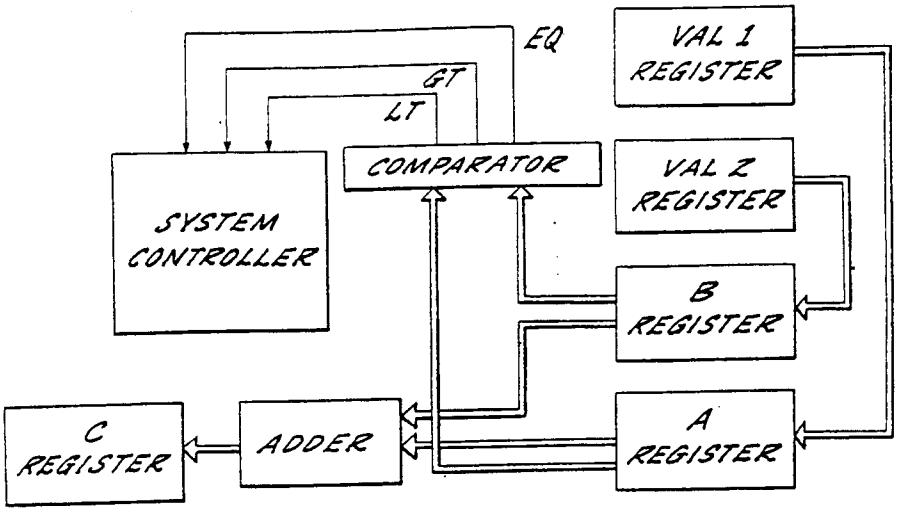

EQ
GT
LT
VAL 1 REGISTER
COMPARATOR
VAL 2 REGISTER
SYSTEM CONTROLLER
B REGISTER
C REGISTER
ADDER
A REGISTER

Fig. 6.

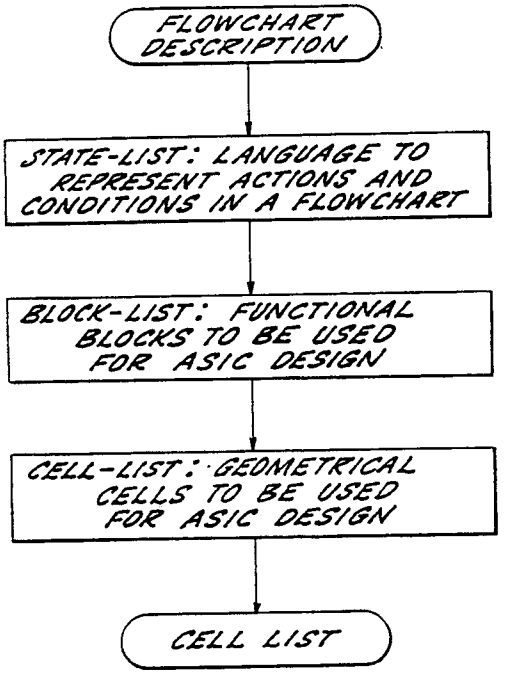

FLOWCHART DESCRIPTION
STATE-LIST: LANGUAGE TO REPRESENT ACTIONS AND CONDITIONS IN A FLOWCHART
BLOCK-LIST: FUNCTIONAL BLOCKS TO BE USED FOR ASIC DESIGN
CELL-LIST: GEOMETRICAL CELLS TO BE USED FOR ASIC DESIGN
CELL LIST

Fig. 9.

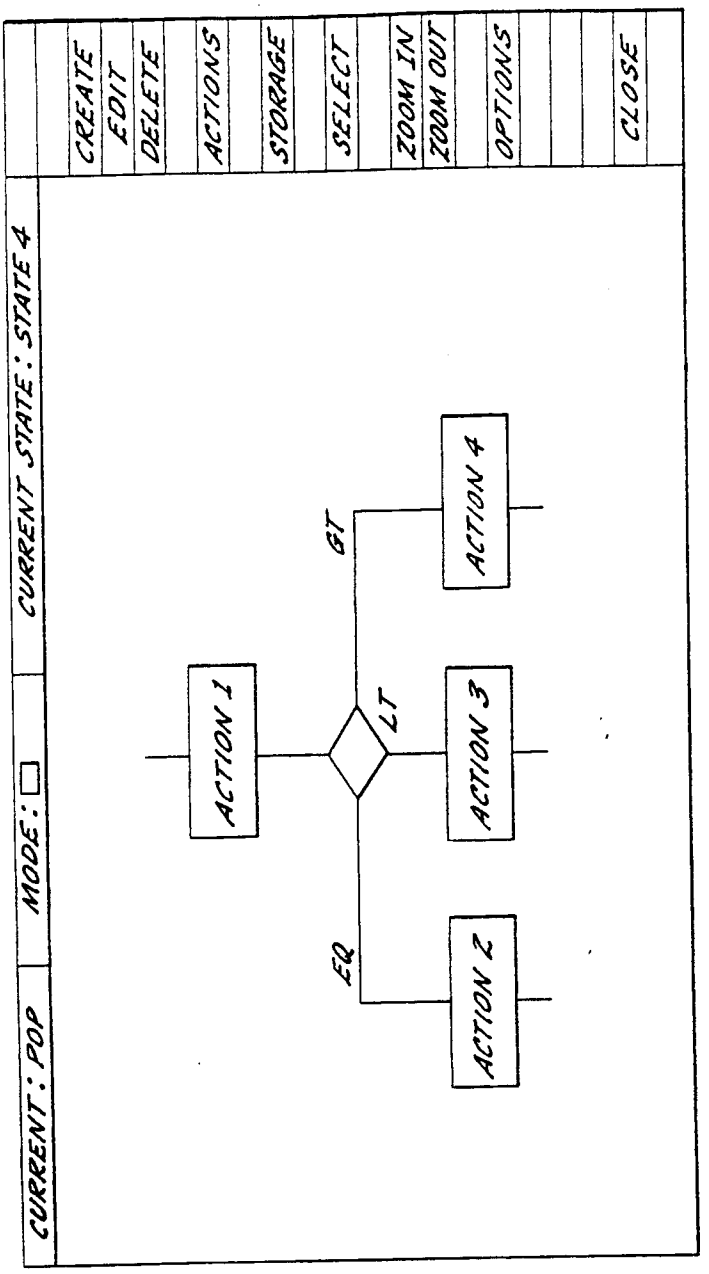

CREATE
EDIT
DELETE
ACTIONS
STORAGE
SELECT
ZOOM IN
ZOOM OUT
OPTIONS
CLOSE
CURRENT STATE: STATE 4
MODE: □
CURRENT: POP
ACTION 1
GT
LT
EQ
ACTION 4
ACTION 3
ACTION 2

FIG. 7.

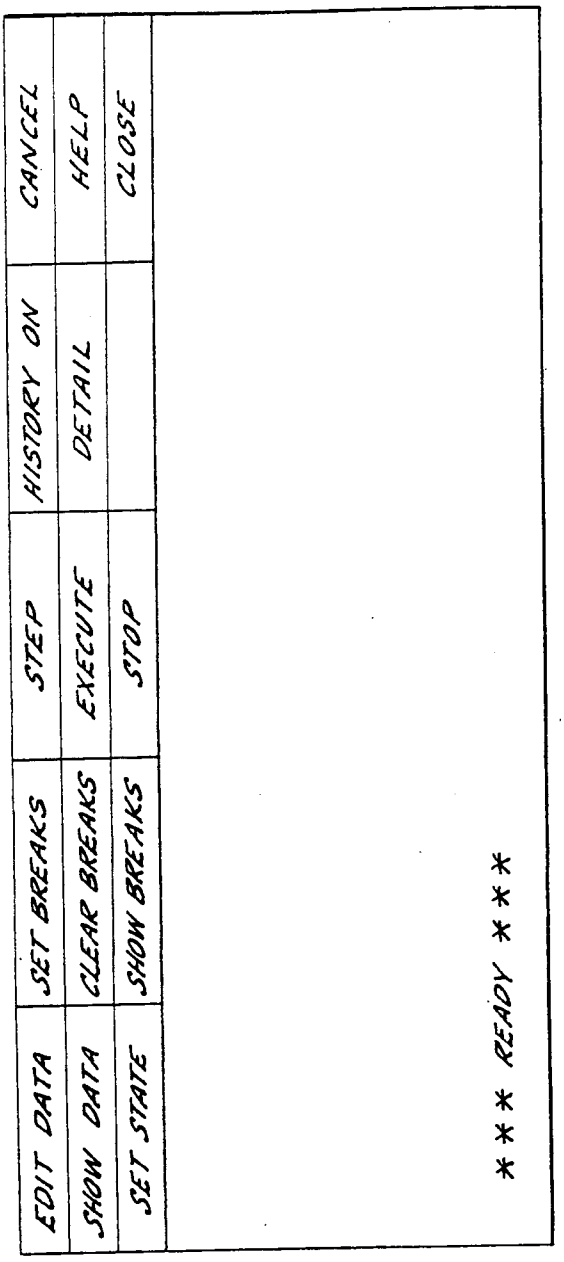

EDIT DATA
SET BREAKS
STEP
HISTORY ON
CANCEL
SHOW DATA
CLEAR BREAKS
EXECUTE
DETAIL
HELP
SET STATE
SHOW BREAKS
STOP
CLOSE
**** READY ****

FIG. 8.

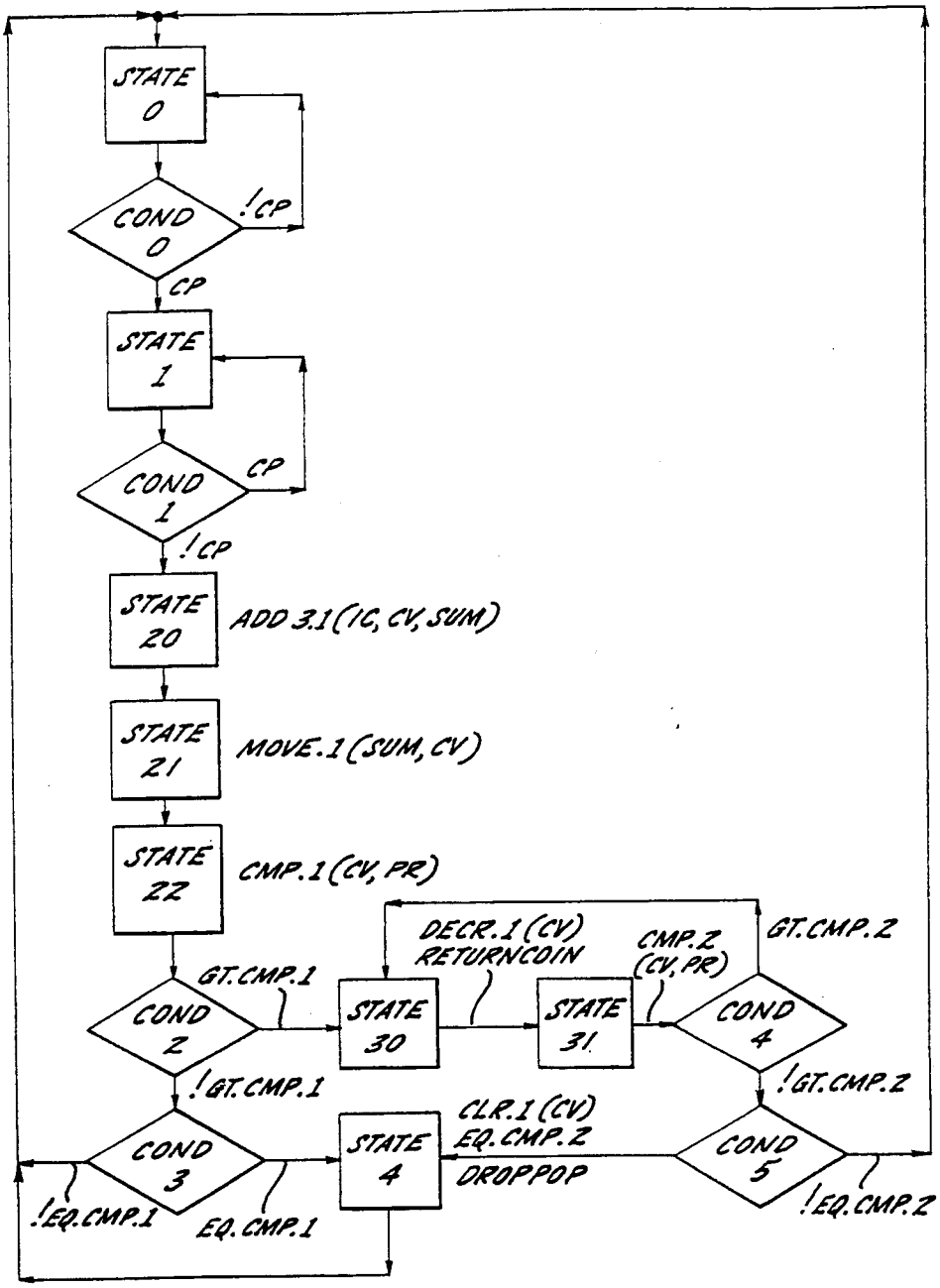

STATE 0
COND 0
!CP
CP
STATE 1
COND 1
CP
!CP
STATE 20
ADD 3.1 (IC, CV, SUM)
STATE 21
MOVE.1 (SUM, CV)
STATE 22
CMP.1 (CV, PR)
GT.CMP.1
COND 2
!GT.CMP.1
STATE 30
DECR.1 (CV)
RETURNCOIN
STATE 31
CMP.2 (CV, PR)
GT.CMP.2
COND 4
!GT.CMP.2
COND 3
!EQ.CMP.1
EQ.CMP.1
STATE 4
CLR.1 (CV)
EQ.CMP.2
DROPPOP
COND 5
!EQ.CMP.2

FIG. 10.

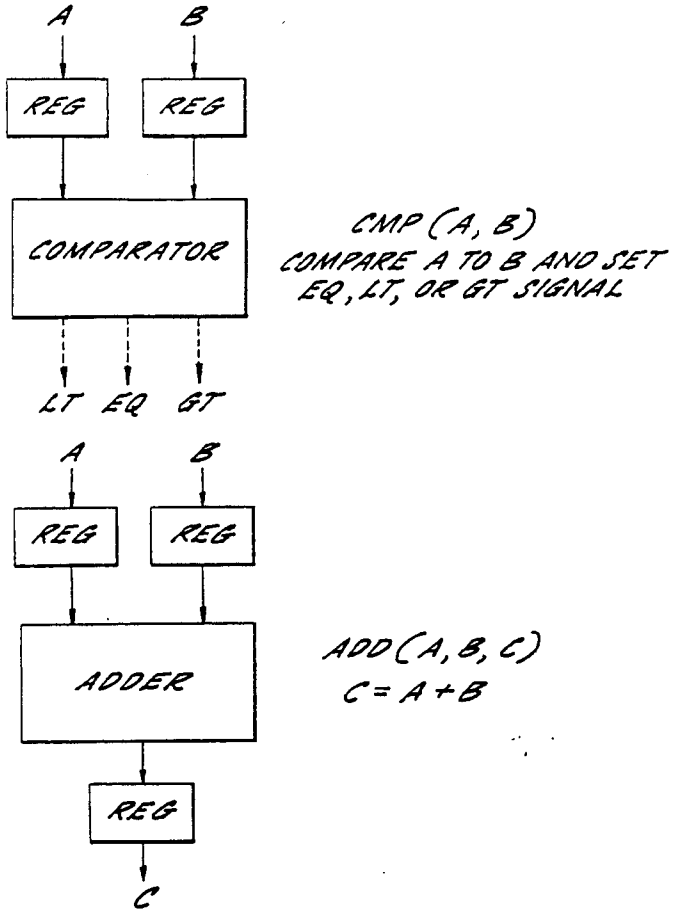

A
B
REG
REG
COMPARATOR
CMP (A, B)
COMPARE A TO B AND SET
EQ, LT, OR GT SIGNAL
LT
EQ
GT
A
B
REG
REG
ADDER
ADD (A, B, C)
C = A + B
REG
C
A
COUNTER
DECR (A)
A = A - 1
A-1
DATA PATH
CONTROL SIGNAL (SIGNAL BIT)

Fig. 11.

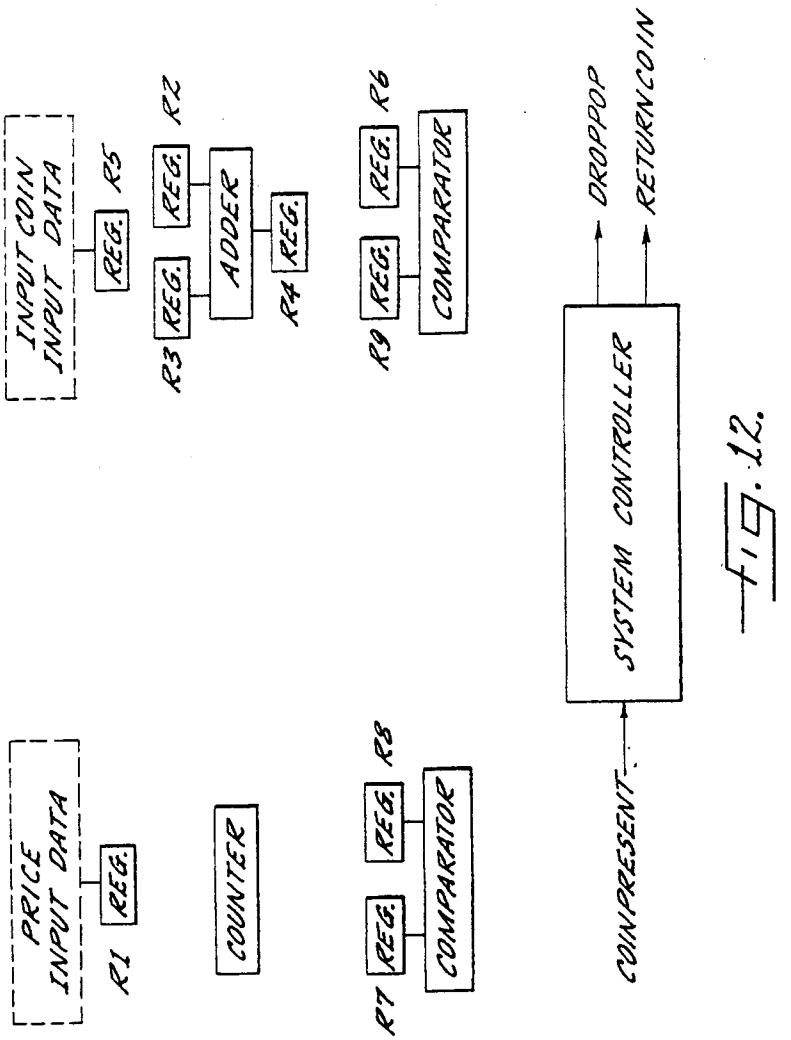

INPUT COIN INPUT DATA
REG.
R5
REG.
R2
REG.
R3
ADDER
R4
REG.
REG.
R6
REG.
R9
COMPARATOR
DROPPOP
RETURN COIN
SYSTEM CONTROLLER
COINPRESENT
PRICE INPUT DATA
REG.
R1
COUNTER
REG.
R8
REG.
R7
COMPARATOR

FIG. 12.

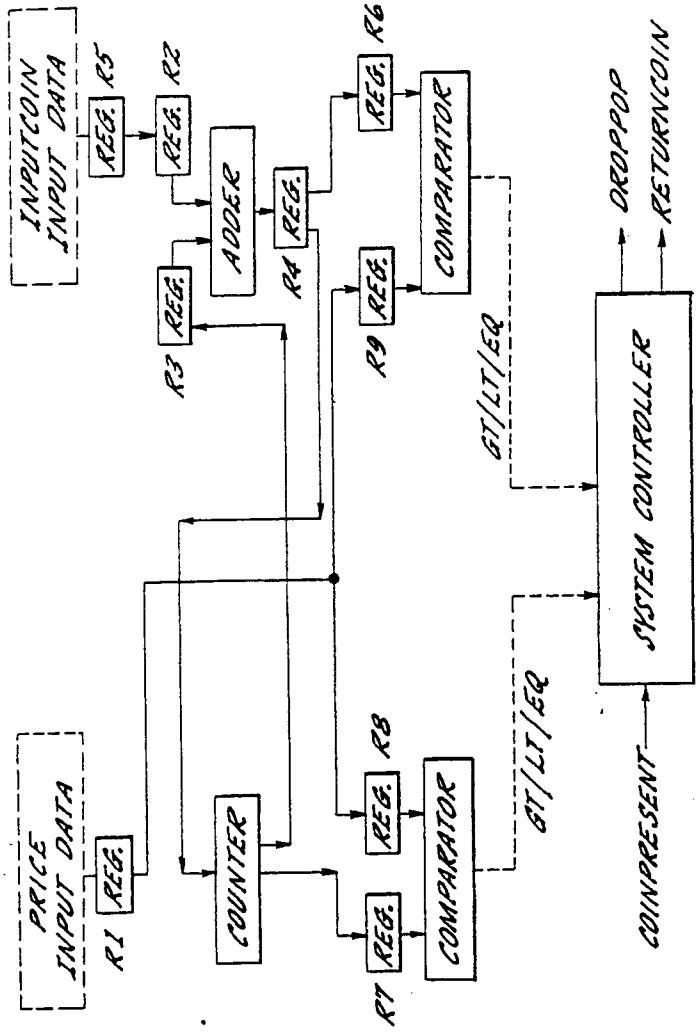

INPUT COIN
INPUT DATA
REG.
R5
REG.
R2
ADDER
REG.
R4
REG.
R6
COMPARATOR
REG.
R3
REG.
R9
GT/LT/EQ
DROP POP
RETURN COIN
SYSTEM CONTROLLER
PRICE
INPUT DATA
REG.
R1
COUNTER
REG.
R8
REG.
R7
COMPARATOR
GT/LT/EQ
COIN PRESENT

FIG. 13.

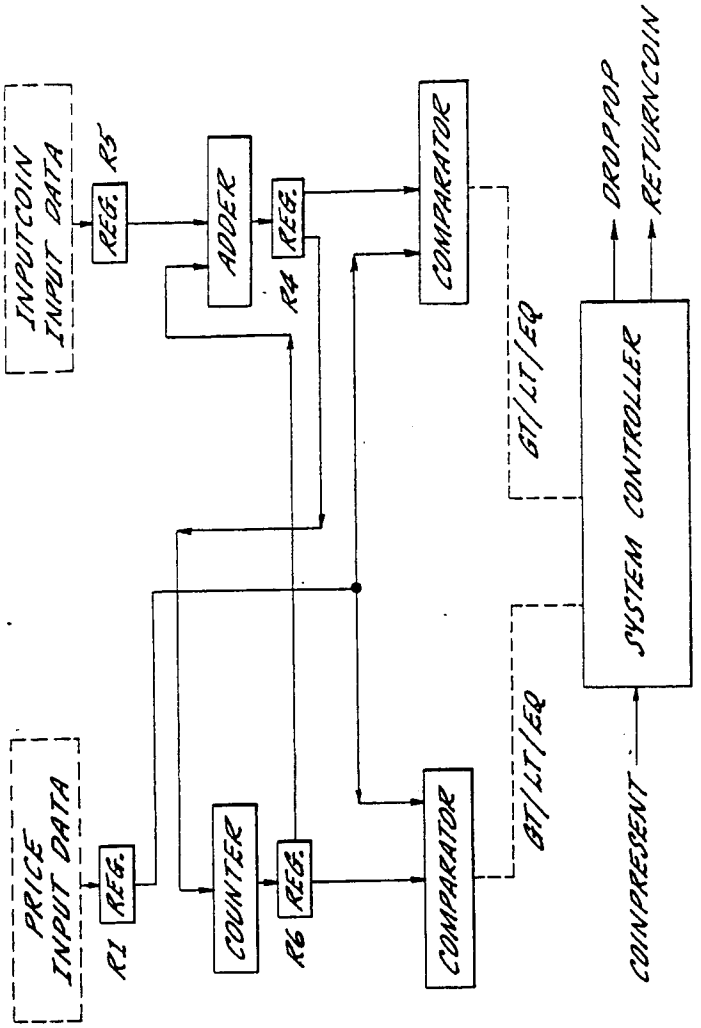

INPUTCOIN INPUT DATA
REG.
R3
ADDER
REG.
R4
COMPARATOR
GT/LT/EQ
DROPPOP
RETURNCOIN
SYSTEM CONTROLLER
PRICE INPUT DATA
REG.
R1
COUNTER
REG.
R6
COMPARATOR
GT/LT/EQ
COINPRESENT

FIG. 14.

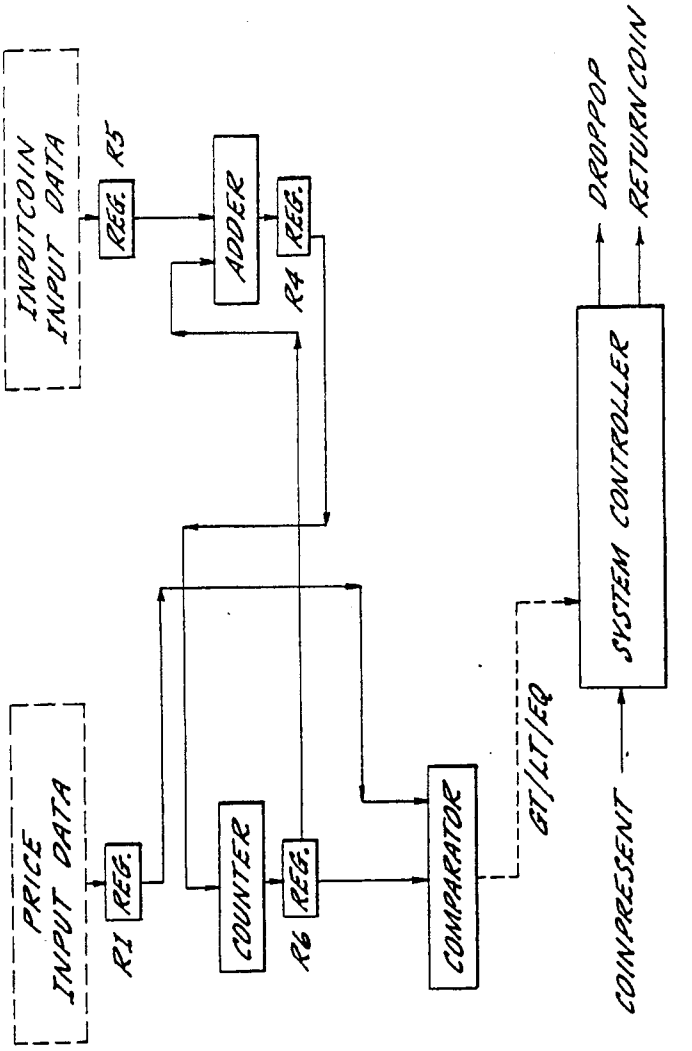

INPUTCOIN
INPUT DATA
REG.
R5
ADDER
REG.
R4
DROPPOP
RETURNCOIN
SYSTEM CONTROLLER
PRICE
INPUT DATA
REG.
R1
COUNTER
REG.
R6
COMPARATOR
GT/LT/EQ
COINPRESENT

Fig. 15.

KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS

FIELD AND BACKGROUND OF THE INVENTION

This invention relates to the design of integrated circuits, and more particularly relates to a computer-aided method and apparatus for designing integrated circuits.

An application specific integrated circuit (ASIC) is an integrated circuit chip designed to perform a specific function, as distinguished from standard, general purpose integrated circuit chips, such as microprocessors, memory chips, etc. A highly skilled design engineer having specialized knowledge in VLSI circuit design is ordinarily required to design a ASIC. In the design process, the VLSI design engineer will consider the particular objectives to be accomplished and tasks to be performed by the integrated circuit and will create structural level design specifications which define the various hardware components required to perform the desired function, as well as the interconnection requirements between these components. A system controller must also be designed for synchronizing the operations of these components. This requires an extensive and all encompassing knowledge of the various hardware components required to achieve the desired objectives, as well as their interconnection requirements, signal level compatibility, timing compatibility, physical layout, etc. At each design step, the designer must do tedious analysis. The design specifications created by the VLSI design engineer may, for example, be in the form of circuit schematics, parameters or specialized hardware description languages (HDLs).

From the structural level design specifications, the description of the hardware components and interconnections is converted to a physical chip layout level description which describes the actual topological characteristics of the integrated circuit chip. This physical chip layout level description provides the mask data needed for fabricating the chip.

Due to the tremendous advances in very large scale integration (VLSI) technology, highly complex circuit systems are being built on a single chip. With their complexity and the demand to design custom chips at a faster rate, in large quantities, and for an ever increasing number of specific applications, computer-aided design (CAD) techniques need to be used. CAD techniques have been used with success in design and verification of integrated circuits, at both the structural level and at the physical layout level. For example, CAD systems have been developed for assisting in converting VLSI structural level descriptions of integrated circuits into the physical layout level topological mask data required for actually producing the chip. Although the presently available computer-aided design systems greatly facilitate the design process, the current practice still requires highly skilled VLSI design engineers to create the necessary structural level hardware descriptions.

There is only a small number of VLSI designers who possess the highly specialized skills needed to create structural level integrated circuit hardware descriptions. Even with the assistance of available VLSI CAD tools, the design process is time consuming and the probability of error is also high because of human involvements. There is a very significant need for a better and more cost effective way to design custom integrated circuits.

SUMMARY OF THE INVENTION

In accordance with the present invention a CAD (computer-aided design) system and method is provided which enables a user to define the functional requirements for a desired target integrated circuit, using an easily understood functional architecture independent level representation, and which generates therefrom the detailed information needed for directly producing an application specific integrated circuit (ASIC) to carry out those specific functions. Thus, the present invention, for the first time, opens the possibility for the design and production of ASICs by designers, engineers and technicians who may not possess the specialized expert knowledge of a highly skilled VLSI design engineer.

The functional architecture independent specifications of the desired ASIC can be defined in a suitable manner, such as in list form or preferably in a flowchart format. The flowchart is a highly effective means of describing a sequence of logical operations, and is well understood by software and hardware designers of varying levels of expertise and training. From the flowchart (or other functional specifications), the system and method of the present invention translates the functional architecture independent specifications into structural an architecture specific level definition of an integrated circuit, which can be used directly to produce the ASIC. The structural level definition includes a list of the integrated circuit hardware cells needed to achieve the functional specifications. These cells are selected from a cell library of previously designed hardware cells of various functions and technical specifications. The system also generates data paths among the selected hardware cells. In addition, the present invention generates a system controller and control paths for the selected integrated circuit hardware cells. The list of hardware cells and their interconnection requirements may be represented in the form of a netlist. From the netlist it is possible using either known manual techniques or existing VLSI CAD layout systems to generate the detailed chip level geometrical information (e.g. mask data) required to produce the particular application specific integrated circuit in chip form.

The preferred embodiment of the system and method of the present invention which is described more fully hereinafter is referred to as a Knowledge Based Silicon Compiler (KBSC). The KBSC is an ASIC design methodology based upon artificial intelligence and expert systems technology. The user interface of KBSC is a flowchart editor which allows the designer to represent VLSI systems in the form of a flowchart. The KBSC utilizes a knowledge based expert system, with a knowledge base extracted from expert ASIC designers with a high level of expertise in VLSI design to generate from the flowchart a netlist which describes the selected hardware cells and their interconnection requirements.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be better understood by reference to the detailed description which follows, taken in connection with the accompanying drawings, in which

FIG. 1*a* illustrates a functional level design representation of a portion of a desired target circuit, shown in the form of a flowchart;

FIG. 1*b* illustrates a structural level design representation of an integrated circuit;

FIG. 1*c* illustrates a design representation of a circuit at a physical layout level, such as would be utilized in the fabrication of an integrated circuit chip;

FIG. 2 is a block schematic diagram showing how integrated circuit mask data is created from flowchart descriptions by the KBSC system of the present invention;

FIG. 3 is a somewhat more detailed schematic illustration showing the primary components of the KBSC system;

FIG. 4 is a schematic illustration showing how the ASIC design system of the present invention draws upon selected predefined integrated circuit hardware cells from a cell library;

FIG. 5 is an example flowchart defining a sequence of functional operations to be performed by an integrated circuit;

FIG. 6 is a structural representation showing the hardware blocks and interconnection requirements for the integrated circuit defined in FIG. 5;

FIG. 7 is an illustration of the flowchart editor window;

FIG. 8 is an illustration of the flowchart simulator window;

FIG. 9 is an illustration of the steps involved in cell list generation;

FIG. 10 is an example flowchart for a vending machine system;

FIG. 11 illustrates the hardware components which correspond to each of the three macros used in the flowchart of FIG. 10;

FIG. 12 is an initial block diagram showing the hardware components for an integrated circuit as defined in the flowchart of FIG. 10;

FIG. 13 is a block diagram corresponding to FIG. 12 showing the interconnections between blocks;

FIG. 14 is a block diagram corresponding to FIG. 13 after register optimization; and

FIG. 15 is a block diagram corresponding to FIG. 14 after further optimization.

DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

FIGS. 1*a*, 1*b* and 1*c* illustrate three different levels of representing the design of an integrated circuit. FIG. 1*a* shows a functional (or behavioral) representation architecture independent in the form of a flowchart. A flowchart is a graphic representation of an algorithm and consists of two kinds of blocks or states, namely actions and conditions (decisions). Actions are conventionally represented in the flowchart by a rectangle or box, and conditions are represented by a diamond. Transitions between actions and conditions are represented by lines with arrows. FIG. 1*b* illustrates a structural (or logic) level representation of an integrated circuit. In this representation, blocks are used to represent integrated architecture specific circuit hardware components for performing various functions, and the lines interconnecting the blocks represent paths for the flow of data or control signals between the blocks. The blocks may, for example, represent hardware components such as adders, comparators, registers, system controllers, etc. FIG. 1*c* illustrates a physical layout level representation of an integrated circuit design, which provides the detailed mask data necessary to actually manufacture the devices and conductors which together comprise integrated circuit.

As noted earlier, the design of an integrated circuit at the structural level requires a design engineer with highly specialized skills and expertise in VLSI design. In the KBSC system of the present invention, however, integrated circuits can be designed at a functional level because the expertise in VLSI design is provided and applied by the invention. Allowing the designer to work with flowcharts instead of logic circuit schematics simplifies the task of designing custom integrated circuits, making it quicker, less expensive and more reliable. The designer deals with an algorithm using simple flowcharts at an architecture independent functional (behavioral) level, and needs to know only the necessary logical steps to complete a task, rather than the specific means for accomplishing the task. Designing with flowcharts requires less work in testing because flowcharts allow the designer to work much closer to the algorithm. On the other hand, previously existing VLSI design tools require the designer to represent an algorithm with complex circuit schematics at a structural level, therefore requiring more work in testing. Circuit schematics make it harder for the designer to cope with the algorithm function which needs to be incorporated into the target design because they intermix the hardware and functional considerations. Using flowcharts to design custom integrated circuits will allow a large number of system designers to access VLSI technology, where previously only a small number of designers had the knowledge and skills to create the necessary structural level hardware descriptions.

The overall system flow is illustrated in FIG. 2. The user enters the functional specifications of the circuit into the knowledge based silicon compiler (KBSC) **10** in the form of a flowchart **11**. The KBSC **10** then generates a netlist **15** from the flowchart. The netlist **15** includes a custom generated system controller, all other hardware cells required to implement the necessary operations, and interconnection information for connecting the hardware cells and the system controller. The netlist can be used as input to any existing VLSI layout and routing tool **16** to create mask data **18** for geometrical layout.

System Overview

The primary elements or modules which comprise the KBSC system are shown in FIG. 3. In the embodiment illustrated and described herein, these elements or modules are in the form of software programs, although persons skilled in the appropriate art will recognize that these elements can easily be embodied in other forms, such as in hardware.

Referring more particularly to FIG. 3, it will be seen that the KBSC system **10** includes a program **20** called EDSIM, which comprises a flowchart editor **21** for creating and editing flowcharts and a flowchart simulator **22** for simulation and verification of flowcharts. Actions to be performed by each of the rectangles represented in the flowchart are selected from a macro library **23**. A program **30** called PSCS (path synthesizer and cell selector) includes a data and control path synthesizer module **31**, which is a knowledge based system for data and control path synthesis. PSCS also includes a cell selector **32** for selecting the cells required for system design. The cell selector **32** selects from a cell

library 34 of previously designed hardware cells the appropriate cell or cells required to perform each action and condition represented in the flowchart. A controller generator 33 generates a custom designed system controller for controlling the operations of the other hardware cells. The knowledge base 35 contains ASIC design expert knowledge required for data path synthesis and cell selection. Thus, with a functional flowchart input, PSCS generates a system controller, selects all other hardware cells, generates data and control paths, and generates a netlist describing all of this design information.

The KBSC system employs a hierarchal cell selection ASIC design approach, as is illustrated in FIG. 4. Rather than generating every required hardware cell from scratch, the system draws upon a cell library 34 of previously designed, tested and proven hardware cells of various types and of various functional capabilities with a given type. The macro library 23 contains a set of macros defining various actions which can be specified in the flowchart. For each macro function in the macro library 23 there may be several hardware cells in the cell library 34 of differing geometry and characteristics capable of performing the specified function. Using a rule based expert system with a knowledge base 35 extracted from expert ASIC designers, the KBSC system selects from the cell library 34 the optimum cell for carrying out the desired function.

Referring again to FIG. 3, the cells selected by the cell selector 32, the controller information generated by the controller generator 33 and the data and control paths generated by the data/control path synthesizer 31 are all utilized by the PSCS program 30 to generate the netlist 15. The netlist is a list which identifies each block in the circuit and the interconnections between the respective inputs and outputs of each block. The netlist provides all the necessary information required to produce the integrated circuit. Computer-aided design systems for cell placement and routing are commercially available which will receive netlist data as input and will lay out the respective cells in the chip, generate the necessary routing, and produce mask data which can be directly used by a chip foundry in the fabrication of integrated circuits.

System Requirements

The KBSC system can be operated on a suitable programed general purpose digital computer. By way of example, one embodiment of the system is operated in a work station environment such as Sun3 and VAXStation-II/GPX Running UNIX Operating System and X Window Manager. The work station requires a minimum of 8 megabytes of main storage and 20 megabytes of hard disk space. The monitor used is a color screen with 8-bit planes. The software uses C programming language and INGRES relational data base.

The human interface is mainly done by the use of pop up menus, buttons, and a special purpose command language. The permanent data of the integrated circuit design are stored in the data base for easy retrieval and update. Main memory stores the next data temporarily, executable code, design data (flowchart, logic, etc.), data base (cell library), and knowledge base. The CPU performs the main tasks of creating and simulating flowcharts and the automatic synthesis of the design.

Flowchart Example

To describe the mapping of a flowchart to a netlist, consider an example flowchart shown in FIG. 5, which is of part of a larger overall system. In this illustrative flowchart, two variables, VAL1 and VAL2 are compared and if they are equal, they are added together. In this instance, the first action (Action 1) involves moving the value of variable VALI to register A. The second action comprises moving the value of variable VAL2 to register B. Condition 1 comprises comparing the values in registers A and B. Action 3 comprises adding the values of registers A and B and storing the result in register C.

In producing an integrated circuit to carry out the function defined in FIG. 5, the KBSC maps the flowchart description of the behavior of the system to interconnection requirements between hardware cells. The hardware cells are controlled by a system controller which generates all control signals. There are two types of variables involved in a system controller:

(1) Input variables: These are generated by hardware cells, and/or are external input to the controller. These correspond to conditions in the flowchart.

(2) Output variables: These are generated by the system controller and correspond to actions in the flowchart.

FIG. 6 illustrates the results of mapping the flowchart of FIG. 5 onto hardware cells. The actions and the conditions in the flowchart are used for cell selection and data and control path synthesis. The VALI register and VAL2 register and the data paths leading therefrom have already been allocated in actions occurring before Action 1 in our example. Action 1 causes generation of the data register A. Similarly, Action 2 causes the allocation of data register B. The comparator is allocated as a result of the comparison operation in Condition 1. The comparison operation is accomplished by (1) selecting a comparator cell, (2) mapping the inputs of the comparator cell to registers A and B, (3) generating data paths to connect the comparator with the registers A and B and (4) generating input variables corresponding to equal to, greater than, and less than for the system controller. Similarly the add operation in Action 3 causes selection of the adder cell, mapping of the adder parameters to the registers and creating the data paths.

Following this methodology, a block list can be generated for a given flowchart. This block list consists of a system controller and as many other blocks as may be required for performing the necessary operations. The blocks are connected with data paths, and the blocks are controlled by the system controller through control paths. These blocks can be mapped to the cells selected from a cell library to produce a cell list.

Interactive Flowchart Editor and Simulator

The creation and verification of the flowchart is the first step in the VLSI design methodology. The translation from an algorithm to an equivalent flowchart is performed with the Flowchart Editor 21 (FIG. 3). The verification of the edited flowchart is performed by the Flowchart Simulator 22 The Flowchart Editor and Simulator are integrated into one working environment for interactive flowchart editing, with a designer friendly interface.

EDSIM is the program which contains the Flowchart Editor 21 and the Flowchart Simulator 22. It also provides functions such as loading and saving flow-

charts. EDSIM will generate an intermediate file, called a statelist, for each flowchart. This file is then used by the PSCS program **30** to generate a netlist.

Flowchart Editor

The Flowchart Editor **21** is a software module used for displaying, creating, and editing the flowchart. This module is controlled through the flowchart editing window illustrated in FIG. 7. Along with editing functions the Flowchart Editor also provides checking of design errors.

The following is a description of the operations of the Flowchart Editor. The main editing functions include, create, edit, and delete states, conditions, and transitions. The create operation allows the designer to add a new state, condition, or transitions to a flowchart. Edit allows the designer to change the position of a state, condition or transition, and delete allows the designer to remove a state, condition or transition from the current flowchart. States which contain actions are represented by boxes, conditions are represented by diamonds, and transitions are represented by lines with arrows showing the direction of the transition.

Edit actions allows the designer to assign actions to each box. These actions are made up of macro names and arguments. An example of arguments is the setting and clearing of external signals. A list of basic macros available in the macro library **23** is shown in Table 1.

TABLE 1

Macro	Description
ADD (A,B,C)	C = A + B
SUB (A,B,C)	C = A − B
MULT (A,B,C)	C = A * B
DIV (A,B,C)	C = A div B
DECR (A)	A = A − 1
INCR (A)	A = A + 1
CLR (A)	A = 0
REG (A,B)	B = A
CMP (A,B)	Compare A to B and set EQ,LT,GT signals
CMP0 (A)	Compare A to 0 and set EQ,LT,GT signals
NEGATE (A)	A = NOT (A)
MOD (A,B,C)	C = A Modulus B
POW (A,B,C)	C = A B
DC2 (A,S1,S2,S3,S4)	Decode A into S1,S2,S3,S4
EC2 (S1,S2,S3,S4,A)	Encode S1,S2,S3,S4 into A
MOVE (A,B)	B = A
CALL sub-flowchart (A,B, . . .)	Call a sub-flowchart. Pass A,B . . .
START (A,B, . . .)	Beginning state of a sub-flowchart
STOP (A,B . . .)	Ending state of a sub-flowchart

The Flowchart Editor also provides a graphical display of the flowchart as the Flowchart Simulator simulates the flowchart. This graphical display consists of boxes, diamonds, and lines as shown in FIG. 7. All are drawn on the screen and look like a traditional flowchart. By displaying the flowchart on the screen during simulation it allows the designer to design and verify the flowchart at the same time.

Flowchart Simulator

The Flowchart Simulator **22** is a software module used for simulating flowcharts. This module is controlled through the simulator window illustrated in FIG. 8. The Flowchart Simulator simulates the transitions between states and conditions in a flowchart. The following is a list of the operations of the Flowchart Simulator:

edit data—Change the value of a register or memory.
set state—Set the next state to be simulated.
set detail or summary display—Display summary or detail information during simulation.
set breaks—Set a breakpoint.
clear breaks—Clear all breakpoints.
show breaks—Display current breakpoints.
step—Step through one transition.
execute—Execute the flowchart.
stop—Stop executing of the flowchart. history ON or history OFF—Set history recording on or off.
cancel—Cancel current operation.
help—Display help screen.
close—Close the simulator window.

The results of the simulation are displayed within the simulator window. Also the editor window will track the flowchart as it is being simulated. This tracking of the flowchart makes it easy to edit the flowchart when an error is found.

Cell Selection

The Cell Selector **32** is a knowledge based system for selecting a set of optimum cells from the cell library **34** to implement a VLSI system. The selection is based on functional descriptions in the flowchart, as specified by the macros assigned to each action represented in the flowchart. The cells selected for implementing a VLSI system depend on factors such as cell function, fabrication technology used, power limitations, time delays etc. The cell selector uses a knowledge base extracted from VLSI design experts to make the cell selection.

To design a VLSI system from a flowchart description of a user application, it is necessary to match the functions in a flowchart with cells from a cell library. This mapping needs the use of artificial intelligence techniques because the cell selection process is complicated and is done on the basis of a number of design parameters and constraints. The concept used for cell selection is analogous to that used in software compilation. In software compilation a number of subroutines are linked from libraries. In the design of VLSI systems, a functional macro can be mapped to library cell.

FIG. **4** illustrates the concept of hierarchical cell selection. The cell selection process is performed in two steps:

(1) selection of functional macros
(2) selection of geometrical cells

A set of basic macros is shown in Table 1. A macro corresponds to an action in the flowchart. As an example, consider the operation of adding A and B and storing the result in C. This function is mapped to the addition macro ADD(X, Y, Z). The flowchart editor and flowchart simulator are used to draw the rectangles, diamonds and lines of the flowchart, to assign a macro selected from the macro library **23** to each action represented in the flowchart, and to verify the functions in flowcharts. The flowchart is converted into an intermediate form (statelist) and input to the Cell Selector.

The Cell Selector uses a rule based expert system to select the appropriate cell or cells to perform each action. If the cell library has a number of cells with different geometries for performing the operation specified by the macro, then an appropriate cell can be selected on the basis of factors such as cell function, process technology used, time delay, power consumption, etc.

The knowledge base of Cell Selector **32** contains information (rules) relating to:

(1) selection of macros
(2) merging two macros

(3) mapping of macros to cells
(4) merging two cells
(5) error diagnostics

The above information is stored in the knowledge base **35** as rules.

Cell List Generation

FIG. **9** shows the cell list generation steps. The first step of cell list generation is the transformation of the flowchart description into a structure that can be used by the Cell Selector. This structure is called the statelist. The blocklist is generated from the statelist by the inference engine. The blocklist contains a list of the functional blocks to be used in the integrated circuit. Rules of the following type are applied during this stage.

map arguments to data paths
map actions to macros
connect these blocks

Rules also provide for optimization and error diagnostics at this level.

The cell selector maps the blocks to cells selected from the cell library **34**. It selects an optimum cell for a block. This involves the formulation of rules for selecting appropriate cells from the cell library. Four types of information are stored for each cell. These are:

(1) functional level information: description of the cell at the register transfer level.

(2) logic level information: description in terms of flip-flops and gates.

(3) circuit level information: description at the transistor level.

(4) Layout level information: geometrical mask level specification.

The attributes of a cell are:

cell name
description
function
width
height
status
technology
minimum delay
typical delay
maximum delay
power
file
designer
date
comment
inspector

In the cell selection process, the above information can be used. Some parameters that can be used to map macros to cells are:

(1) name of macro
(2) function to be performed
(3) complexity of the chip
(4) fabrication technology
(5) delay time allowed
(6) power consumption
(7) bit size of macro data paths

Netlist Generation

The netlist is generated after the cells have been selected by PSCS. PSCS also uses the macro definitions for connecting the cell terminals to other cells. PSCS uses the state-to-state transition information from an intermediate form representation of a flowchart (i.e. the statelist) to generate a netlist. PSCS contains the following knowledge for netlist generation:

(1) Data path synthesis
(2) Data path optimization
(3) Macro definitions
(4) Cell library
(5) Error detection and correction

The above information is stored in the knowledge base **35** as rules. Knowledge engineers help in the formulation of these rules from ASIC design experts. The macro library **23** and the cell library **34** are stored in a database of KBSC.

A number of operations are performed by PSCS. The following is a top level description of PSCS operations:

(1) Read the flowchart intermediate file and build a statelist.
(2) current_context=START
(3) Start the inference engine and load the current context rules.
(4) Perform one of the following operations depending upon current_context:
(a) Modify the statelist for correct implementation.
(b) Create blocklist, macrolist and data paths.
(c) Optimize blocklist and datapath list and perform error checks.
(d) Convert blocks to cells.
(e) Optimize cell list and perform error checks.
(f) Generate netlist.
(g) Optimize netlist and perform error checks and upon completion Goto 7.
(5) If current_context has changed, load new context rules.
(6) Goto **4**.
(7) Output netlist file and stf files and Stop.

In the following sections, operations mentioned in step 4 are described. The Rule Language and PSCS display are also described.

Rule Language

The rule language of PSCS is designed to be declarative and to facilitate rule editing. In order to make the expert understand the structure of the knowledge base, the rule language provides means for knowledge representation. This will enable the format of data structures to be stated in the rule base, which will enable the expert to refer to them and understand the various structures used by the system. For example, the expert can analyze the structure of wire and determine its components. The expert can then refer these components into rules. If a new object has to be defined, then the expert can declare a new structure and modify some existing structure to link to this new structure. In this way, the growth of the data structures can be visualized better by the expert. This in turn helps the designer to update and append rules.

The following features are included in the rule language:

(i) Knowledge representation in the form of a record structure.
(ii) Conditional expressions in the antecedent of a rule.
(iii) Facility to create and destroy structure in rule actions.
(iv) The assignment statement in the action of a rule.
(v) Facility for input and output in rule actions.
(vi) Provide facility to invoke C functions from rule actions.

The rule format to be used is as follows:

The rule format to be used is as follows:

Rule	<number> <context>	
If {		
	<if-clause>	
}		
Then	{	
	<then-clause>	
}		
where	<number>	rule number
	<context>	context in which this rule is active
	<if-clause>	the condition part of the rule
	<then-clause>	the action part of the rule

Inference Strategy

The inference strategy is based on a fast pattern matching algorithm. The rules are stored in a network and the requirement to iterate through the rules is avoided. This speeds up the execution. The conflict resolution strategy to be used is based on the following:

(1) The rule containing the most recent data is selected.

(2) The rule which has the most complex condition is selected.

(3) The rule declared first is selected.

Rule Editor

PSCS provides an interactive rule editor which enables the expert to update the rule set. The rules are stored in a database so that editing capabilities of the database package can be used for rule editing. To perform this operation the expert needs to be familiar with the various knowledge structures and the inferencing process. If this is not possible, then the help of a knowledge engineer is needed.

PSCS provides a menu from which various options can be set. Mechanisms are provided for setting various debugging flags and display options, and for the overall control of PSCS.

Facility is provided to save and display the blocklist created by the user. The blocklist configuration created by the user can be saved in a file and later be printed with a plotter. Also the PSCS display can be reset to restart the display process.

PSCS Example Rules:

Rule	1	
	IF	no blocks exist
	THEN	generate a system controller.
Rule	2	
	IF	a state exists which has a macro AND this macro has not been mapped to a block
	THEN	find a corresponding macro in the library and generate a block for this macro.
Rule	3	
	IF	there is a transition between two states AND there are macros in these states using the same argument
	THEN	make a connection from a register corresponding to the first macro to another register corresponding to the second macro.
Rule	4	
	IF	a register has only a single connection from another register
	THEN	combine these registers into a single register.
Rule	5	
	IF	there are two comparators AND input data widths are of the same size AND one input of these is same AND the outputs of the comparators are used to perform the same operation.
	THEN	combine these comparators into a single comparator.
Rule	6	
	IF	there is a data without a register
	THEN	allocate a register for this data.
Rule	7	
	IF	all the blocks have been interconnected AND a block has a few terminals not connected
	THEN	remove the block and its terminals, or issue an error message.
Rule	8	
	IF	memory is to be used, but a block has not been created for it
	THEN	create a memory block with data, address, read and write data and control terminals.
Rule	9	
	IF	a register has a single connection to a counter
	THEN	combine the register and the counter; remove the register and its terminals.
Rule	10	
	IF	there are connections to a terminal of a block from many different blocks
	THEN	insert a multiplexor; remove the connections to the terminals and connect them to the input of the multiplexor; connect the output of the multiplexor to the input of the block.

Additional rules address the following points:

remove cell(s) that can be replaced by using the outputs of other cell(s)

reduce multiplexor trees

use fan-out from the cells, etc.

Soft Drink Vending Machine Controller Design Example

The following example illustrates how the previously described features of the present invention are employed in the design of an application specific integrated circuit (ASIC). In this illustrative example the ASIC is designed for use as a vending machine controller. The vending machine controller receives a signal each time a coin has been deposited in a coin receiver. The coin value is recorded and when coins totalling the correct amount are received, the controller generates a signal to dispense a soft drink. When coins totalling more than the cost of the soft drink are received, the controller dispenses change in the correct amount.

This vending machine controller example is patterned after a textbook example used in teaching digital system controller design. See Fletcher, William I., *An Engineering Approach to Digital Design*, Prentice-Hall, Inc., pp. 491-505. Reference may be made to this textbook example for a more complete explanation of this vending machine controller requirements, and for an understanding and appreciation of the complex design procedures prior to the present invention for designing the hardware components for a controller.

FIG. 10 illustrates a flowchart for the vending machine controller system. This flowchart would be entered into the KBSC system by the user through the flowchart editor. Briefly reviewing the flowchart, the controller receives a coin present signal when a coin is received in the coin receiver. State0 and cond0 define a waiting state awaiting deposit of a coin. The symbol CP represents "coin present" and the symbol !CP repre-

sents "coin not present". Statel and condI determine when the coin has cleared the coin receiver. At state20, after receipt of a coin, the macro instruction ADD3.1 (lc, cv, sum) instructs the system to add lc (last coin) and cv (coin value) and store the result as sum. The macro instruction associated with state21 moves the value in the register sum to cv. The macro CMP.1 at state22 compares the value of cv with PR (price of soft drink) and returns signals EQ, GT and LT. The condition cond2 tests the result of the compare operation CMP.1. If the result is "not greater than" (!GT.CMP.1), then the condition cond3 tests to see whether the result is "equal" (EQ.CMP.1). If the result is "not equal" (!EQ.CMP.1), then control is returned to state0 awaiting the deposit of another coin. If cond3 is EQ, then state4 generates a control signal to dispense a soft drink (droppop) and the macro instruction CLR.1(cv) resets cv to zero awaiting another customer.

If the total coins deposited exceed the price, then state30 produces the action "returncoin". Additionally, the macro DECR.1 (cv) reduces the value of cv by the amount of the returned coin. At state31 cv and PR are again compared. If cv is still greater than PR, then control passes to state30 for return of another coin. The condition cond5 tests whether the result of CMP.2 is EQ and will result in either dispensing a drink (droppop) true or branching to state0 awaiting deposit of another coin. The macros associated with the states shown in FIG. 10 correspond to those defined in Table 1 above and define the particular actions which are to be performed at the respective states.

Appendix A shows the intermediate file or "statelist" produced from the flowchart of FIG. 10. This statelist is produced as output from the EDSIM program 20 and is used as input to the PSCS program 30 (FIG. 3).

FIG. 11 illustrates for each of the macros used in the flowchart of FIG. 10, the corresponding hardware blocks. It will be seen that the comparison macro CMP (A,B) results in the generation of a register for storing value A, a register for storing value B, and a comparator block and also produces control paths to the system controller for the EQ, LT, and GT signals generated as a result of the comparison operation. The addition macro ADD (A,B,C) results in the generation of a register for each of the input values A and B, a register for the output value C, and in the generation of an adder block. The macro DECR (A) results in the generation of a counter block. The PSCS program 30 maps each of the macros used in the flowchart of FIG. 10 to the corresponding hardware components results in the generation of the hardware blocks shown in FIG. 12. In generating the illustrated blocks, the PSCS program 30 relied upon rules 1 and 2 of the above listed example rules.

FIG. 13 illustrates the interconnection of the block of FIG. 12 with data paths and control paths. Rule 3 was used by the data/control path synthesizer program 31 in mapping the data and control paths.

FIG. 14 shows the result of optimizing the circuit by applying rule 4 to eliminate redundant registers. As a result of application of this rule, the registers R2, R3, R7, R8, and R9 in FIG. 13 were removed. FIG. 15 shows the block diagram after further optimization in which redundant comparators are consolidated. This optimization is achieved in the PSCS program 30 by application of rule 5.

Having now defined the system controller block, the other necessary hardware blocks and the data and control paths for the integrated circuit, the PSCS program 30 now generates a netlist 15 defining these hardware components and their interconnection requirements. From this netlist the mask data for producing the integrated circuit can be directly produced using available VLSI CAD tools.

```
name rpop;
data path @ic<0:5>, cv<0:5>, sum<0:5>, @pr<0:5>;
{
state4 : state0;
state30 : state31;
state21 : state22;
state20 : state21;
state0 :. !cp state0;
state0 :. cp state1;
state1 :. cp state1;
state1 :. !cp state20;
state22 :. GT.CMP.1 state30;
state22 :. !GT.CMP.1*EQ.CMP.1 state4;
state22 :. !GT.CMP.1*!EQ.CMP.1 state0;
state31 :. GT.CMP.2 state30;
state31 :. !GT.CMP.2*EQ.CMP.2 state4;
state31 :. !GT.CMP.2*!EQ.CMP.2 state0;
state30 :: returncoin;
state30 :: DECR.1(cv);
state4 :: droppop;
state4 :: CLR.1(cv);
state31 :: CMP.2(cv,pr);
state22 :: CMP.1(cv,pr);
state21 :: MOVE.1(sum,cv);
state20 :: ADD3.1(ic,cv,sum);
}
```

That which I claimed is:

1. A computer-aided design system for designing an application specific integrated circuit directly from architecture independent functional specifications for the integrated circuit, comprising

a macro library defining a set of architecture independent operations comprised of actions and conditions;

input specification means operable by a user for defining architecture independent functional specifications for the integrated circuit, said functional specifications being comprised of a series of operations comprised of actions and conditions, said input specification means including means to permit the user to specify for each operation a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing the available operations defined in said macro library;

cell selection means for selecting from said cell library for each macro specified by said input specification means, appropriate hardware cells for performing the operation defined by the specified macro, said cell selection means comprising an expert system including a knowledge base containing rules for. selecting hardware cells from said cell library and inference engine means for selecting appropriate hardware cells from said cell library in accordance with the rules of said knowledge base; and

netlist generator means cooperating with said cell selection means for generating as output from the system a netlist defining the hardware cells which are needed to achieve the functional requirements of the integrated circuit and the connections therebetween.

2. The system as defined in claim 1 wherein said input means comprises means specification for receiving user

input of a list defining the series of actions and conditions.

3. The system as defined in claim 1 additionally including mask data generator means for generating from said netlist the mask data required to produce an integrated circuit having the specified functional requirements.

4. The system as defined in claim 1 wherein said input means comprises flowchart editor means specification for creating a flowchart having elements representing said series of actions and conditions.

5. The system as defined in claim 4 additionally including flowchart simulator means for simulating the functions defined in the flowchart to enable the user to verify the operation of the integrated circuit.

6. The system as defined in claim 1 additionally including data path generator means cooperating with said cell selection means for generating data paths for the hardware cells selected by said cell selection means.

7. The system as defined in claim 6 wherein said data path generator means comprises a knowledge base containing rules for selecting data paths between hardware cells and inference engine means for selecting data paths between the hardware cells selected by said cell selection means in accordance with the rules of said knowledge base and the arguments of the specified macros.

8. The system as defined in claim 6 additionally including control generator means for generating a controller and control paths for the hardware cells selected by said cell selection means.

9. A computer-aided design system for designing an application specific integrated circuit directly from a flowchart defining architecture independent functional requirements of the integrated circuit comprising

a marco library defining a set of architecture independent operations comprised of actions and conditions;

flowchart editor means operable by a user for creating a flowchart having elements representing said architecture independent operations;

said flowchart editor means including macro specification means for permitting the user to specify for each operation represented in the flowchart a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing the available operations defined in said macro library;

cell selection means for selecting form said cell library for each specified macro, appropriate hardware cells for performing the operation defined by the specified macro, said cell selection means comprising an expert system including a knowledge base containing rules for selecting hardware cells from said cell library and inference engine means for selecting appropriate hardware cells from said cell library in accordance with the rules of said knowledge base; and

data path generator means cooperating with said cell selection means for generating data paths for the hardware cells selected by said cell selector means, said data path generator means comprising a knowledge base containing rules for selecting data paths between hardware cells and inference engine means for selecting data paths between hardware cells selected by said cell selection means in accordance with the rules of said knowledge base and the arguments of the specified macros.

10. The system as defined in claim 9 additionally including control generator means for generating a controller and control paths for the hardware cells selected by said cell selection means.

11. A computer-aided design system for designing an application specific integrated circuit directly from a flowchart defining architecture independent functional requirements of the integrated circuit, comprising

flowchart editor means operable by a user for creating a flowchart having boxes representing architecture independent actions, diamonds representing architecture independent conditions, and lines with arrows representing transitions between actions and condition and including means for specifying for each box or diamond, a particular action or condition to be performed;

a cell library defining a set of available integrated circuit hardware cells for performing actions and conditions;

a knowledge base containing rules for selecting hardware cells from said cell library and for generating data and control paths for hardware cells; and

expert system means operable with said knowledge base for translating the flowchart defined by said flowchart editor means into a netlist defining the necessary hardware cells and data and control paths required in an integrated circuit having the specified functional requirements.

12. The system as defined in claim 11 including mask data generator means for generating from said netlist the mask data required to produce an integrated circuit having the specified functional requirements.

13. A computer-aided design process for designing an application specific integrated circuit which will perform a desired function comprising

storing a set of definitions of architecture independent actions and conditions;

storing data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set;

storing in an expert system knowledge base a set of rules for selecting hardware cells to perform the actions and conditions;

describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions;

specifying for each described action and condition of the series one of said stored definitions which corresponds to the desired action or condition to be performed; and

selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit, said step of selecting a hardware cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert system knowledge base and generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.

14. A process as defined in claim 13, including generating from the netlist the mask data required to produce an integrated circuit having the desired function.

15. A process as defined in claim 13 including the further step of generating data paths for the selected integrated circuit hardware cells.

16. A process as defined in claim 15 wherein said step of generating data paths comprises applying to the selected cells a set of data path rules stored in a knowledge base and generating the data paths therefrom.

17. A process as defined in claim 16 including the further step of generating control paths for the selected integrated circuit hardware cells.

18. A knowledge based design process for designing an application specific integrated circuit which will perform a desired function comprising

storing in a macro library a set of macros defining architecture independent actions and conditions;

storing in a cell library a set of available integrated circuit hardware cells for performing the actions and conditions;

storing in a knowledge base set of rules for selecting hardware cells from said cell library to perform the actions and conditions defined by the stored macros;

describing for a proposed application specific integrated circuit a flowchart comprised of elements representing a series of architecture independent actions and conditions which carry out the function to be performed by the integrated circuit;

specifying for each described action and condition of said series a macro selected from the macro library which corresponds to the action or condition; and

applying rules of said knowledge base to the specified macros to select from said cell library the hardware cells required for performing the desired function of the application specific integrated circuit and generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.

19. A process as defined in claim 18 also including the steps of

storing in said knowledge base a set of rules for creating data paths between hardware cells, and

applying rules of said knowledge base to the specified means to create data paths for the selected hardware cells.

20. A process as defined in claim 19 also including the steps of generating a controller and generating control paths for the selected hardware cells.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432 Page 1 of 4

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

ON TITLE PAGE: under the section "References Cited" under "Other Publications":

"Verifying Compiled Silicon", by E. K. cheng, VLSI Design, Oct. 1984, pp. 1-4." should be -- "Verifying Compiled Silicon", by E. K. Cheng, VLSI Design, Oct. 1984, pp. 1-4." --.

"quality of Designs from An Automatic Logic Generator", by T. D. Friedman et al., IEEE 7th DA Conference, 1970, pp. 71-89." should be -- "Quality of Designs from An Automatic Logic Generator", by T. D. Friedman et al., IEEE 7th DA Conference, 1970, pp. 71-89. --.

"Trevillyan-Trickey, H., Flamel: *A High Level Hardward Compiler*, IEEE Transactions On Computer Aided Design, Mar. 1987, pp. 259-269." should be -- Trevillyan-Trickey, H., Flamel: *A High Level Hardware Compiler*, IEEE Transactions On Computer Aided Design, Mar. 1987, pp. 259-269. --.

In the abstract:

Every occurrence of "functional architecture independent" should be -- architecture independent functional --.

Column 1, line 19, "a" should be -- an --.

[redacted]

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 2, line 10, "functional architecture independent" should be -- architecture independent functional --.

Column 2, line 21, "functional architecture independent" should be -- architecture independent functional --.

Column 2, lines 29-30, "functional architecture independent" should be -- architecture independent functional --.

Column 2, line 31, "structural" should be after "specific".

Column 3, lines 51-52, "representation" should be after "architecture independent".

Column 3, lines 61-62, "integrated" should be after "specific".

Column 6, line 62, after "22" insert -- . --.

Column 7, line 43 (in Table 1), "C = A B" should be -- C = A^B --.

Column 8, line 9 should end with the word "flowchart" and "history" should begin on the next line.

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432 Page 3 of 4

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 10, line 23, "data paths" should be -- datapaths --.

Column 10, line 68, delete "The rule format to be used is as follows:".

Column 12, line 54, "Enqineering" should be -- Engineering --.

Column 13, line 55, "block" should be -- blocks --.

In the Claims:

Column 14, line 68, before "means" (first occurrence) insert -- specification --; after "means" (second occurrence) delete "specification".

Column 15, line 9, before "means" (first occurrence) insert -- specification --; after "means" (second occurrence) delete "specification".

Column 15, line 35, after "circuit" insert -- , --.

Column 15, line 36, "marco" should be -- macro --.

Column 15, line 49, "form" should be -- from --.

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 16, line 14, "condition" should be
-- conditions --.

Column 17, line 19, after "base" insert -- a --.

Signed and Sealed this

Fourteenth Day of January, 1992

Attest:

Attesting Officer

HARRY F. MANBECK, JR.

Commissioner of Patents and Trademarks

EXHIBIT 2

Case 5:03-cv-04669-JW Document 238-2 Filed 08/27/2004 Page 2 of 25

Exhibit A ('432 Patent)

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
13. A computer-aided design process for designing an application specific integrated circuit which will perform a desired function comprising	During manufacture of a desired application specific integrated circuit (ASIC) chip that is designed to perform a specific purpose, a process of designing the desired ASIC using a computer, the process comprising: ("application specific integrated circuit (ASIC)"= an integrated circuit chip designed to perform a specific function.) <u>Support</u> '432 Patent: Column 1, lines 13-17; column 2, lines 15-20. RCL000207-223. "computer-aided design": The use of computers to aid in design layout and analysis. IEEE Standard Dictionary of Electrical and Electronic Terms,	A. "A computer-aided design process for designing" -- a process that uses a computer for designing, as distinguished from a computer-aided manufacturing process, which uses a computer to direct and control the manufacturing process. *Intrinsic Evidence:* **'432 patent:** 1:9-12; 16:34-65 [Attachment 2[1], DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 16-18[2]; **CAD Tool Integration For ASIC Design:** at 364-365 [Attachment 21, KBSC000031-KBSC000036]. *Dictionaries, Treatises, Textbooks, etc.:* **IBM Dictionary of Computing:** at 129-130 [Attachment 16, DEF083932-DEF084703]; **IC Mask Design:** at 1-24 Attachment 17,

[1] Referenced attachments are attached to this Exhibit A.

[2] "Decl. Kowalski ¶ ¶" refers to the pertinent paragraphs in the Declaration and Summary of Opinions of Dr. Thaddeus J. Kowalski on Construction of Disputed Claim Terms of United States Patent No. 4,922,432 that support Synopsys' and Defendants' proposed constructions and oppose Ricoh's proposed constructions for the disputed claim term, phrase or clause. Ricoh objects to the citation of extrinsic evidence by Synopsys and the Aeroflex defendants, and reserves its right to seek discovery regarding any such extrinsic evidence and respond with its own extrinsic evidence.

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	Fourth Edition (1988). RCL011382-388 at RCL011384. "function": A specific purpose of an entity or its characteristic action. IEEE Standard Dictionary of Electrical and Electronic Terms, Fourth Edition (1988). RCL011382-388 at RCL011386. "function": Any of a group of related actions contributing to a larger action. Merriam-Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011400.	DEF085303-DEF085329]; **Microchip Fabrication:** at 76-82, 274-278 [Attachment 18, DEF085330-085343]; **IEEE Standard Dictionary of Electrical and Electronics Terms:** at 180 [Attachment 19, DEF085299-085302]; **Webster's Dictionary:** at 343, 725 [Attachment 20, DEF085290-DEF085298]. B. "application specific integrated circuit" -- an interconnected miniaturized electronic circuit on a single piece of semiconductor material designed to perform a specific function, as distinguished from standard, general purpose integrated circuits, such as microprocessors, memory chips, etc. *Intrinsic Evidence:* **'432 patent:** 1:13-17; 16:34-65 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 19-20; **CAD Tool Integration For ASIC Design:** at 363 [Attachment 21, KBSC000031-KBSC000036].

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
storing a set of definitions of architecture independent actions and conditions;	Placing in computer memory a library of definitions of the different architecture independent actions and conditions that can be selected for use in the desired ASIC, where the architecture independent actions and conditions do not imply any structure or implementing technology. ("architecture independent action and condition"= functional or behavioral aspects of a portion of a circuit (or circuit segment) that does not imply any set architecture, structure or implementing technology.) <u>Support</u> '432 Patent: Figs. 1a, 1b, 1c, 4; column 2, lines 6-20; column 2, lines 27-34; column 3, line 49 to column 4, line 4; column 4, lines 5-19; column 5, lines 20-22; column 7, lines 24-50; column 8, lines 23-51; column 10, lines 10-12; column 13, lines 2-31.	C.[3] "actions and conditions"-- are the logical steps and decisions that are represented as rectangles and diamonds in the flowchart; collectively logical operations. *Intrinsic Evidence:* **'432 patent:** 2:24-27; 3:50-59; 4:15-19; 4:61-63; 6:3-14; 7:20-23; 8:47-51; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 9, 11; October 1989 Examiner Interview Summary; November 1989 Amendment at 6-7 [Attachment 3, DEF011820-DEF012110]. *Extrinsic Evidence:* **Decl. Kowalski** ¶ 21. *Dictionaries, Treatises, Textbooks, etc.:* **IBM Dictionary of Computing:** at 479 [Attachment 16, DEF083932-DEF084703].

[3] Synopsys and the Defendants disagree with Ricoh's definition of "storing" on this step and the next two steps as "placing in computer memory." Storing means placing on any storage device "that is accessible by the processor for the computer system." IBM Dictionary of Computing at 654 (Attachment 16, DEF083932-DEF084703).

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	"action": A thing done. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011391. "architecture": A unifying or coherent form or structure. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011393. "independent": Not dependent; not requiring or relying on something else. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011394. "condition": Something essential to the appearance or occurrence of something else. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011405A	D. "architecture independent"[4] -- not including (i.e., excluding) a register transfer level (RTL) description or any other description that is hardware architecture dependent. An RTL description consists of: 1) defining the inputs, outputs, and any registers of the proposed ASIC; and, 2) describing for a single clock cycle of the ASIC how the ASIC outputs and any registers are set according to the values of the ASIC inputs and the previous values of the registers; an RTL description defines any control needed for the ASIC. *Intrinsic Evidence:* **'432 patent file history:** April 1989 Amendment at 8-10, 13; November 1989 Amendment at 7 [Attachment 3, DEF011820-DEF012110]; **'435 patent:** Fig. 4; 4:26-32; 5:27-35 [Attachment 4, DEF012503-DEF012518]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 22-26; **Computer Aided VLSI Design Vol.1 No. 4:** at 388

[4] This phrase was not in the original patent application and therefore violates the prohibition of Section 132 of 35 U.S.C. against adding new matter. Moreover, it is indefinite and inadequately described in the '432 patent. The only description is in the '432 patent's file history, which is in the negative.

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		[Attachment 22, KBSC001109-KBSC001131]. E. "a set of definitions of architecture independent actions and conditions" -- a set of named descriptions defining the functionality and arguments for the available logical steps and decisions that may be specified in the flowchart; and excluding a register transfer level description. *Intrinsic Evidence:* **'432 patent:** 4:61-63; 5:20-22; 6:3-14; 7:25-50; 8:47-51; 16:34-65 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 27-28.
storing data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set;	Placing in computer memory a library of cell information that describe hardware cells capable of performing the different architecture independent actions and conditions placed in the library of definitions. ("hardware cells"= previously designed circuit components or structure that have specific physical and functional characteristics used as building blocks for implementing an ASIC to be	F. "hardware cells" -- logic blocks for which the functional level (e.g., register transfer level), logic level (e.g., flip flop and gate level), circuit level (e.g., transistor level), and layout level (e.g., geometrical mask level) descriptions are all defined. *Intrinsic Evidence:* **'432 patent:** 2:34-39; 3:59-67; 5:15-20; 9:24-51; 16:34-65; 16:66-68 [Attachment 2, DEF012564-

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	manufactured.) Support '432 Patent: Fig. 4; column 2, lines 36-39; column 4, line 68 to column 5, line 3; column 5, lines 15-20; column 5, lines 22-25; column 9, lines 24-60; column 10, lines 10-12.	DEF012585]; **'016 patent:** 11:47-13:63; 14:3-22 [Attachment 5, DEF017265-DEF017284]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 29-30; **Computer Aided VLSI Design Vol.1 No. 4:** at 380 [Attachment 22, KBSC001109-KBSC001131]. G. "data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set" -- a set of named integrated circuit hardware cells that includes at least one hardware cell for each stored definition that may be specified for the available logical steps and decisions; where each named hardware cell has corresponding descriptions at the functional level (e.g., register transfer level), logic level (e.g., flip-flop and gate level), circuit level (e.g., transistor level), and layout level (e.g., geometrical mask level) that are all defined. *Intrinsic Evidence:* **'432 patent:** Fig. 4; 2:34-39; 3:59-67; 5:15-20; 5:23-25; 9:24-51; 16:34-65; 16:66-68 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:*

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Decl. Kowalski ¶¶ 31-32.
storing in an expert system knowledge base a set of rules for selecting hardware cells to perform the actions and conditions;	Placing in an expert system knowledge base, that uses a computer memory, a plurality of rules for selecting among the hardware cells placed in the hardware cell library, wherein the rules comprise the expert knowledge of highly skilled VLSI designers formulated as prescribed procedures. ("expert system knowledge base"= database used to store expert knowledge of highly skilled VLSI designers.) ("rules"= the expert knowledge of highly skilled VLSI designers formulated as prescribed procedures.) Support '432 Patent: Column 5, lines 6-8; column 8, line 65 to column 9, line 5; column 9, lines 14-20; column 10, lines 1-10; column 10, line 40 to column 11, line 14; column 11, lines 16-26; column 11, lines 30-32; column 11, line 46 to column 12, line 35. RCL000229-237.	H. "expert system" -- software executing on a computer system that attempts to embody the knowledge of a human expert in a particular field and then uses that knowledge to simulate the reasoning of such an expert to solve problems in that field. This system is comprised of a knowledge base containing rules, working memory containing the problem description, and an inference engine. It solves problems through the selective application of the rules in the knowledge base to the problem description, as distinguished from conventional software, which uses a predefined step-by-step procedure (algorithm) to solve problems. *Intrinsic evidence:* **'432 patent:** 2:58-63; 5:6-8; 8:58-60; 10:39-11:26; 14:50-59; 15:53-58; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'432 patent's file history:** April 1989 Amendment at 9-11, 15, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 7, 9 [Attachment 3, DEF011820-012110]; **'016 patent:** 2:65-3:8 [Attachment 5, DEF017265-DEF017284]; **'435 patent**: 7:32-9:35[Attachment 4, DEF012503-

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	"rule": A prescribed guide for conduct or action; an accepted procedure, custom or habit. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011405. "expert system": (1977): computer software that attempts to mimic the reasoning of a human specialist. Merriam-Webster's Collegiate Dictionary Tenth Edition (1999). RCL011408-410 at RCL011410.	DEF012518]; **'603 patent:** 1:44-49 [Attachment 6, DEF017456-DEF017482]; **An Overview of Logic Synthesis Systems:** at 170 [Attachment 7, DEF011962-DEF011968]; **The CMU Design Automation System:** at 75-77 [Attachment 8, DEF012045-DEF012052]. I. "Knowledge base" -- the portion of the expert system containing a set of rules embodying the expert knowledge for the particular field. *Intrinsic evidence:* **'432 patent:** 10:39-11:15; 14:50-59; 15:53-58; 16:34-65 [Attachment 2, DEF012864-DEF012585]; **'432 patent file history:** April 1989 Amendment at 9-11, 15, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 7, 9 [Attachment 3, DEF011820-DEF012110]; **'435 patent**: 7:32-9:35 [Attachment 4, DEF012503-DEF012518]. J. "a set of rules for selecting hardware cells to perform the actions and conditions" -- a set of rules, each having an antecedent portion (IF) and a consequent portion (THEN), embodying the knowledge of expert designers for application specific integrated circuits, which enables the

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		expert system to map the specified stored definitions for each logical step and decision represented in the flowchart to a corresponding stored hardware cell description. *Intrinsic evidence:* **'432 patent:** 2:58-63; 8:20-30; 8:58-9:62; 10:39-11:26; 14:50-59; 15:53-58; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 9-11, 15, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 7, 9 [Attachment 3, DEF011820-DEF012110]; **'435 patent**: 7:32-9:35 [Attachment 4, DEF012503-DEF012518]; **An Overview of Logic Synthesis Systems:** at 170 [Attachment 7, DEF011962-DEF011968]; **The CMU Design Automation System:** at 75-77 [Attachment 8, DEF012045-DEF012052]. *Extrinsic Evidence for all of the above phrases (H, I, & J) in this step:* **Decl. Kowalski** ¶¶ 33-45; **Computer Aided VLSI Design Vol.1 No. 4:** 351, 377-381, 383, 388-389 [Attachment 22, KBSC001109-KBSC001131]; **'669 patent:** Abstract, 1:13-16, 2:28-33, 4:31-62; 5:21-38, 5:58-68, 6:1-7:68, 17:62 [Attachment 23, DEF 080579-DEF080605]; **The VLSI**

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Design Automation Assistant: at 889-90 [Attachment 28, DEF018660-DEF018664]; **A Rule-Based Logic Circuit Synthesis System for CMOS Gate Arrays**: at 597 [Attachment 29, DEF018277-DEF018283]. *Dictionaries, Treatises, Textbooks, etc. for all of the above phrases (H, I, & J in this step):* **Understanding Expert Systems:** 7-10, 29-30, 40, 42, 74-78, 99-110 [Attachment 24, DEF079512-DEF079741]; **An Artificial Intelligence Approach To VLSI Design:** at 9-15 [Attachment 25, DEF078425-DEF078455]; **Artificial Intelligence Terminology:** at 6 [algorithm], 10 [antecedent], 53 [consequent], 86-87 [expert system], 127 [inference engine], 140 [knowledge based system], 204 [production system], 223 [rule base, rule-based system], 281 [working memory] [Attachment 26, DEF079212-DEF079511]; **Microsoft Press Computer Dictionary:** at 136 [expert system] [Attachment 27, DEF083323-DEF083325]; **IBM Dictionary of Computing:** 591 [rule interpreter, rule-based system] [Attachment 16, DEF083932-DEF084703]; **Expert Systems: A Non-Programmer's Guide:** 8-10, 16 [Attachment 30, DEF082264-DEF082528]; **Expert Systems:**

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Tools and Applications: at 269 [Attachment 31, DEF079985-DEF080283]; **Expert Systems: Principles and Case Studies:** at 11-12 [Attachment 32, DEF079753-DEF079984]; **Knowledge-Based Systems: The View in 1986:** at 16 [Attachment 33, DEF083251-DEF083284].
describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions;	A user describing an input specification containing the desired functions to be performed by the desired ASIC. Support '432 Patent: Column 1, line 13 to column 2, line 3, column 2, lines 6-20, column 2, lines 21-24; column 2, lines 27-34; column 6, lines 58-60; column 13, lines 32-35; column 14, lines 7-29. RCL000207-223, RCL000229-237. "describe": To represent or give an account in words <~ a picture>; to represent by a figure, model, or picture: delineate. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011398. "description": A descriptive statement or account.	K. "describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions" -- the designer represents a sequence of logical steps (rectangles) and decisions (diamonds), and the transitions (lines with arrows) between them in a flowchart format for a proposed application specific integrated circuit. *Intrinsic evidence:* **'432 patent:** Figs. 1a, 5, &7; 2:21-27; 3:20-22; 3:50-59; 4:5-22; 4:35-38; 7:12-23; 16:34-65 [Attachment 2, DEF012564-012585]; **'432 patent file history:** April 1989 Amendment at 9, 11; October 1989 Examiner Interview Summary; November 1989 Amendment at 6-7 [Attachment 3, DEF011820-DEF012110]; **'016 patent:** 7:33-9:52 [Attachment 5, DEF017265-DEF017284]; **'435 patent:** 4:26-33 [Attachment 4, DEF012503-DEF012518]; **Flamel: A High-**

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011398.	**Level Hardware Compiler:** 260-261 [Attachment 9, DEF012034-DEF012044]; **An Overview of Logic Synthesis Systems:** at 168-170 [Attachment 7, DEF011962-DEF011968]; **Methods Used in an Automatic Logic Design Generator (ALERT):** at 595 [Attachment 10, DEF011969-DEF011989]; **Quality of Designs From an Automatic Logic Generator (ALERT):** at 71 [Attachment 11, DEF012005-DEF012023]; **The CMU Design Automation System:** at 73-74 [Attachment 8, DEF012045-DEF012052]; **A New Look at Logic Synthesis:** at 544 [Attachment 12, DEF012024-DEF012030]; **Experiments in Logic Synthesis:** at 235 [Attachment 13, DEF011990-DEF011994]; **CAD Systems for IC Design:** at 3, 7 [Attachment 14, DEF011951-DEF011961]; **Verifying Compiled Silicon:** at 2 [Attachment 15, DEF011948-DEF011950]; **'442 patent:** 5:3-46, [Attachment 35, DEF012392-DEF012401; '603 patent: 2:41-61 [Attachment 6, DEF012392-DEF012401]. *Extrinsic Evidence:* **Decl. Kowalski ¶¶** 46-49; **Computer Aided VLSI Design Vol.1 No. 4:** 351, 377-381, 383, 388-389 [Attachment 22, KBSC001109-KBSC001131];

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Silicon Compilation: at 48-49 [Attachment 34, DEF076335-DEF076341]. *Dictionaries, Treatises, Textbooks, etc.:* **Webster's Ninth New Collegiate Dictionary** at 343, 1074, 1073 [Attachment 20, DEF085290-DEF085298].
specifying for each described action and condition of the series one of said stored definitions which corresponds to the desired action or condition to be performed; and	Specifying for each desired function to be performed by the desired ASIC one of the definitions of the architecture independent actions and conditions stored in the library of definitions that is associated with the desired function. ("specifying"= mapping or associating a desired function to be performed by the manufactured ASIC with a definition from the library of definitions.) <u>Support</u> '432 Patent: Column 5, lines 20-22; column 7, lines 24-50; column 8, lines 23-51; column 8, lines 65-67; column 9, lines 8-18; column 10, lines 10-12; column 13, lines 2-31.	L. "specifying for each described action and condition of the series one of said stored definitions" -- the designer assigns one definition from the set of stored definitions for each of the described logical steps and decisions represented in the flowchart. *Intrinsic Evidence:* **'432 patent:** Fig. 5; 3:20-22; 4:61-63; 5:20-22; 7:24-25; 8:23-26; 8:51-56; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'016 patent:** 6:12-32 [Attachment 5, DEF017265-DEF017284]. *Dictionaries, Treatises, Textbooks, etc.* **Webster's Ninth New Collegiate Dictionary** at 1132 [Attachment 20, DEF085290-DEF085298]. M. "which corresponds to the desired action or condition to be performed" -- each specified definition must correspond to the intended step or

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		decision to be performed. *Intrinsic Evidence:* **'432 patent:** Fig. 5; 3:20-22; 4:61-63; 5:20-22; 7:24-25; 8:23-26; 8:51-56; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'016 patent:** 6:12-32 [Attachment 5, DEF017265-DEF017284]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 50-52. *Dictionaries, Treatises, Textbooks, etc.:* **Webster's Ninth New Collegiate Dictionary** at 110 [Attachment 20, DEF085290-DEF085298].
selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit, said step of selecting a hardware	Selecting from the plurality of hardware cells in the hardware cell library a hardware cell for performing the desired function of the desired ASIC through application of the rules; and generating a netlist that identifies the hardware cells needed to perform the function of the desired ASIC and the necessary parameters for connecting the respective inputs and outputs of each hardware cell, the netlist is passed to the next subsequent step in the process for manufacturing the desired ASIC. ("netlist"= a description of the hardware components	Synopsys and Defendants provide their constructions for the disputed claim terms, phrases, and clauses and support for these two separate "selecting" and "generating for the selected integrated hardware cells" steps below and separately for each step. Synopsys and Defendants dispute Ricoh's attempt to improperly and misleadingly combine these two separate steps.

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert system knowledge base and generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.	(and their interconnections) needed to manufacture the ASIC as used by subsequent processes, e.g., mask development, foundry, etc.) Support '432 Patent: Figs. 4, 9, 13; column 1, lines 17-26; column 2, lines 34-36; column 2, lines 42-44; column 5, lines 25-29; column 5, lines 35-40; column 8, lines 21-23; column 8, lines 26-41; column 8, lines 58-64; column 9, lines 8-24; column 9, line 64 to column 10, line 7; column 10, lines 13-34; column 13, line 36 to column 14, line 6. RCL000207-223, RCL000229-237. "expert system": (1977): computer software that attempts to mimic the reasoning of a human specialist. Merriam-Webster's Collegiate Dictionary Tenth Edition (1999). RCL011408-410 at RCL011410. "interconnection": To connect with one another. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011403.	

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	"requirement": Something wanted or needed-Necessity. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011404.	
selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit, said step of selecting a hardware cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert	See the "selecting" row at column 2, *supra*, for Ricoh's analysis of this claim element. Because the selecting step was amended to include (and was granted as including) both "applying" and "generating" substeps, Ricoh has interpreted this claim element as a single claim element. Ricoh believes that Synopsys and the ASIC Defendants' attempt to cull out the "generating" substep as a separate and distinct claim element is an improper rewriting of the claim.	N. "selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit" -- mapping the specified stored definitions for each logical step and decision represented in the flowchart to a corresponding stored hardware cell description. *Intrinsic Evidence:* **'432 patent:** Fig. 4; 3:16-19; 4:66-5:3; 5:22-29; 8:31-37; 8:58-60; 9:52-60; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 10 [Attachment 3, DEF011820-DEF012110]. *Extrinsic Evidence:* **Decl. Kowalski** ¶ 55. O. "said step of selecting a hardware cell comprising applying to the specified definition of the action

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
system knowledge base and		or condition to be performed, a set of cell selection rules stored in said expert system knowledge base" -- the mapping of the specified definitions to the stored hardware cell descriptions must be performed by an expert system having an inference engine for selectively applying a set of rules, each rule having an antecedent portion (IF) and a consequent portion (THEN), embodying the knowledge of expert designers for application specific integrated circuits, which enables the expert system to map the specified stored definitions for each logical step and decision represented in the flowchart to a corresponding stored hardware cell description. *Intrinsic Evidence:* **'432 patent:** Abstract; 2:58-63; 5:6-8; 8:29-37; 8:58-60; 9:8-13; 11:16-26; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 8-11, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 4, 6-7, 9 [¶¶ Attachment 3, DEF011820-DEF012110]; **'435 patent**: 7:32-9:35[Attachment 4, DEF012503-DEF012518]; **'603 patent:** at 3:59-63 [Attachment 6, DEF017456-DF017482]; **'016 patent:** 3:5-8; 9:67-10:2, [Attachment 5,

Case 5:03-cv-04669-JW Document 238-2 Filed 08/27/2004 Page 19 of 25

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		DEF017265-DEF017284; **An Overview of Logic Synthesis Systems:** at 170 [Attachment 7, DEF011962-DEF011968]; **The CMU Design Automation System:** at 75-77 [Attachment 8 , DEF012045-DEF012052]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 56-57 Same as set forth for phrases H, I, & J above. *Dictionaries, Treatises, Textbooks, etc.* Same as set forth for phrases H, I & J above.

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.	See the "selecting" row at column 2, *supra*, for Ricoh's analysis of this claim element. Because the selecting step was amended to include (and was granted as including) both "applying" and "generating" substeps, Ricoh has interpreted this claim element as a single claim element. Ricoh believes that Synopsys and the ASIC Defendants' attempt to cull out the "generating" substep as a separate and distinct claim element is an improper rewriting of the claim.	P. "Netlist" -- a structural description that includes a custom controller type hardware cell and all other hardware cells required to implement the application specific integrated circuit's operations and any necessary interconnections including the necessary control and data path information for connecting the hardware cells and the controller. *Intrinsic Evidence:* **'432 patent:** Abstract; 1:17-37; 2:39-44; 4:39-43; 5:8-12; 5:30-40; 9:62-10:9; 12:31-35; 13:55-14:3; 16:34-65 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 58-60. Q. "generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit" -- producing a list of the needed hardware cells by eliminating any mapped hardware cells that are redundant or otherwise unnecessary and producing a custom controller type hardware cell for providing the needed control for those other hardware cells and

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Intrinsic Evidence: **'432 patent:** Abstract; 1:17-37; 2:39-44; 4:39-43; 5:8-12; 5:30-40; 9:62-10:9; 13:55-14:3; 16:34-65 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 61-62. R. "generating …interconnection requirements therefor" -- producing the necessary structural control paths and data paths for the needed hardware cells and the custom controller. *Intrinsic Evidence:* **'432 patent:** Abstract; Figs. 6 & 13-15; 1:17-37; 2:39-44; 3:23-25; 3:40-45; 4:39-43; 5:8-12; 5:30-40; 9:62-10:9; 13:55-14:3; 16:34-65 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 63-64.
14. A process as defined in claim 13, including generating from the netlist the mask data required to	The process of claim 13, including producing from the netlist of hardware cells to be included in the designed ASIC mask data which can be directly used by a chip foundry in the fabrication of the ASIC.	S. "generating from the netlist the mask data required to produce an integrated circuit having the desired function" -- producing, from the structural netlist, the detailed layout level geometrical information required for

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
produce an integrated circuit having the desired function.	Support '432 Patent: Figs. 1c, 2; column 1, lines 42-43; column 2, lines 44-49; column 3, line 68 to column 4, line 4; column 4, lines 44-46; column 5, lines 40-46.	manufacturing the set of photomasks that are used by the processes that directly manufacture the application specific integrated circuit. *Intrinsic Evidence:* **'432 patent:** Abstract; Fig. 1c; 1:42-44; 1:54-58; 2:44-49; 4:44-46; 5:40-46; 14:4-6; 16:34-68 [Attachment 2, DEF012564-DEF012585]; **'016 patent:** 1:41-47; 1:57-61; 4:11-13 [Attachment 5, DEF017265-DEF017284]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 65-66.
15. A process as defined in claim 13 including the further step of generating data paths for the selected integrated circuit hardware cells.	The process of claim 13, including producing signal lines for carrying data to the hardware cells. Support '432 Patent: Figs. 11, 13; column 2, lines 39-40; column 3, lines 60-65. RCL000207-223.	T. "generating data paths for the selected integrated circuit hardware cells" -- producing the necessary structural descriptions of the data paths for the mapped hardware cells. *Intrinsic Evidence:* **'432 patent:** Abstract; 2:39-40; 4:63-66; 5:6-12; 5:30-37; 6:29-31; 6:37-43; 6:50-53; 9:62-10:9; 13:55-14:3; 16:34-65; 17:1-3 [Attachment 2, DEF012564-DEF012585]; **'016 patent:** 2:21-25 [Attachment 5, DEF017265-017284]. *Extrinsic Evidence:*

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Decl. Kowalski ¶¶ 67-69. *Dictionaries, Treatises, Textbooks, etc.:* **IEEE Standard Dictionary** at 898 [Attachment 19, DEF085344-DEF085347].
16. A process as defined in claim 15 wherein said step of generating data paths comprises applying to the selected cells a set of data path rules stored in a knowledge base and generating the data paths therefrom.	The process of claim 15, wherein the step of producing signal lines for carrying data comprises applying rules, which are placed in computer memory, to produce the signal lines for carrying data to the hardware cells. Support '432 Patent: Column 4, lines 64-66. RCL000207-223.	U. "said step of generating data paths comprises applying to the selected cells a set of data path rules stored in a knowledge base and generating the data paths therefrom" -- the generating step must be performed by at least an expert system having an inference engine for selectively applying a set of rules, each having an antecedent portion (IF) and a consequent portion (THEN), embodying the knowledge of expert designers for application specific integrated circuits, which enables the expert system to produce the necessary data paths for the mapped hardware cells. *Intrinsic Evidence:* **'432 patent:** Abstract; 5:6-12; 9:62-10:9; 13:55-14:3; 16:34-65; 17:1-7 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 11 [Attachment 3, DEF011820-DEF012110]; **'435 patent**: 7:32-9:35 [Attachment 4, DEF012503-DEF012518]; **An Overview of Logic Synthesis**

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Systems: at 170 [Attachment 7, DEF011962-DEF011968]; **The CMU Design Automation System:** at 75-77 [Attachment 8, DEF012045-DEF012052. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 69-70. Same as set forth for phrases H, I, & J above. *Dictionaries, Treatises, Textbooks, etc.* Same as set forth for phrases H, I & J above.
17. A process as defined in claim 16 including the further step of generating control paths for the selected integrated circuit hardware cells.	The process of claim 16, including producing signal lines for carrying control signals to the hardware cells. Support '432 Patent: Figs. 11, 13; column 2, lines 40-42; column 3, lines 60-65. RCL000207-223.	V. "generating control paths for the selected integrated circuit hardware cells" --producing the necessary structural descriptions of the control paths for the selected hardware cells. *Intrinsic Evidence:* **'432 patent:** Abstract; Figs. 1b & 13-15; 1:17-37; 2:40-42; 3:59-65; 4:39-43; 4:63-65; 5:3-12; 5:30-36; 6:18-27; 11:49-51; 13:51-14:3; 16:34-65; 17:1-10 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 8 [Attachment 3, DEF011820-DEF012110]; **'016 patent:** 2:20-24 [Attachment 5, DEF017265-DEF017284].

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Extrinsic Evidence: **Decl. Kowalski** ¶¶ 71-72. *Dictionaries, Treatises, Textbooks, etc.:* **IEEE Standard Dictionary** at 898 [signal line], [Attachment 19, DEF085344-DEF085347].

EXHIBIT 3

An American National Standard

Acknowledged as An American National Standard
July 8, 1988

IEEE Standard Dictionary of Electrical and Electronics Terms

Fourth Edition

U.S. EPA Headquarters Library
Mail code 3201
1200 Pennsylvania Avenue NW
Washington DC 20460

RCL011382

Library of Congress Catalog Number 88-082198

ISBN: 1-55937-000-9

© Copyright 1988

The Institute of Electrical and Electronics Engineers, Inc

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

November 3, 1988

SH12070

RCL011383

Case 5:03-cv-04669-JW Document 238-3 Filed 08/27/2004 Page 4 of 8

compressor-stator-blade-control system (gas turbines). A means by which the turbine compressor stator blades are adjusted by vary the operating characteristics of the compressor. *See:* **speed-governing system (gas turbines).** 58

computation. *See:* **implicit computation.**

computer (1)(emergency and standby power analog computers). (A) A machine for carrying out calculations. (B) By extension, a machine for carrying out specified transformations on information. 512, 9

(2) (software). (A) A functional unit that can perform substantial computation, including numerous arithmetic operations, or logic operations without intervention by a human operator during a run. (B) A functional programmable unit that consists of one or more associated processing units and peripheral equipment, that is controlled by internally stored programs, and that can perform substantial computation, including numerous arithmetic operations or logic operations, without human intervention. *See:* **programs.** 434

computer-aided design (CAD)(computer applications). The use of computers to aid in design layout and analysis. May included modeling, analysis, simulation, or optimization of designs for production. Often used in combinations such as CAD/CAM. *See:* **computer-aided engineering; computer-aided manufacturing.** 571

computer-aided engineering (CAE)(computer applications). The use of computers to aid in engineering analysis and design. May included solution of mathematical problems, process control, numerical control, and execution of programs performing complex or repetitive calculations. *See:* **computer-aided design; computer-aided manufacturing.** 571

computer-aided inspection (CAI)(computer applications). The use of computers to inspect manufactured parts. 571

computer-aided instruction (CAI)(computer applications). The use of computers to present instructional material and to accept and evaluate student responses. *See:* **computer-based instruction.** 571

computer-aided management (CAM)(computer applications). The application of computers to business management activities. For example, database management, control reporting, and information retrieval. *See:* **decision support system; management information system.** 571

computer-aided manufacturing (CAM)(computer applications). The use of computers and numerical control equipment to aid in manufacturing processes. May include robotics, automation of testing, management functions, control, and product assembly. Often used in combinations such as CAD/CAM. *See:* **computer-aided design; computer-aided engineering.** 571

computer-aided typesetting (computer applications). The use of computers at any stage of the document composition process. This may involve text formatting, input from a word processing system, or computer-aided page makeup. 571

computer-assisted tester (test, measurement and diagnostic equipment). A test not directly programmed by a computer but which operates in association with a computer by using some arithmetic functions of the computer. 54

computer-based instruction (CBI)(computer applications). The use of computers to support any process involving human learning. 571

computer code. A machine code for a specific computer. 255,77

computer component (analog computers). Any part, assembly, or subdivision of a computer, such as resistor, amplifier, power supply, or rack. 9

computer conferencing (computer applications). A form of teleconferencing that allows one or more users to exchange messages on a computer network. 571

computer control (physical process) (electric power systems). A mode of control wherein a computer, using as input the process variables, produces outputs that control the process. *See:* **power system.** 200

computer-control state (analog computers). One of several distinct and selectable conditions of the computer-control circuits. *See:* **balance check; hold; operate; potentiometer set; reset; static test.** 9

computer data (software). Data available for communication between or within computer equipment. Such data can be external (in computer-readable form) or resident within the computer equipment and can be in the form of analog or digital signals. *See:* **computer.** 434

computer diagram (analog computers). A functional drawing showing interconnections between computing elements, such interconnections being specified for the solution of a particular set of equations. *See:* **computer program; problem board.** 9

computer equation (machine equation) (analog computers). An equation derived from a mathematical model for use on a computer which is equivalent or proportional to the original equation. *See:* **scale factor.** 9

computer instruction. A machine instruction for a specific computer. 255, 77

computer-integrated manufacturing (CIM)(computer applications). Use of an integrated system of computer-controlled manufacturing centers. The centers may use robotics, design automation or CAD/CAM (computer-aided design/computer-aided manufacturing technologies. 571

computer interface equipment (surge withstand capability). A device which interconnects a protective relay system to an independent computer, for example, an analog to digital converter, a scanner, a buffer amplifier. 90

computer-managed instruction (CMI)(computer applications). The use of computers for management of student progress. Activities may include record keeping, progress evaluation, and lesson assignment. *See:* **computer-based instruction.** 571

computer network (1) (general). A complex consisting of two or more interconnected computing units. 255, 77

axis that are essentially vertical and horizontal (perpendicular to direction of travel). *See:* feed tube. 52

floating speed (process control). In single-speed or multiple-speed floating control systems, the rate of change of the manipulated variable. 56

floating zero (numerically controlled machines). A characteristic of a numerical machine control permitting the zero reference point on an axis to be established readily at any point in the travel. *Note:* The control retains no information on the location of any previously established zeros. *See:* zero offset. 224,207

float storage (gyro). The sum of attitude storage and the torque command storage in a rate integrating gyro. *See:* attitude storage; torque command storage. 46

float switch (liquid-level switch) (industrial control). A switch in which actuation of the contacts is effected when a a float reaches a predetermined level. *See:* switch. 244,206

flood (charge-storage tubes) (verb). To direct a large-area flow of electrons, containing no spatially distributed information, toward a storage assembly. *Note:* A large-area flow of electrons with spatially distributed information is used in image-converter tubes. *See:* charge-storage tube. 174

floodlight (illuminating engineering). A projector designed for lighting a scene or object to a brightness considerably greater than its surroundings. It usually is capable of being pointed in any direction and is of weatherproof construction. *Note:* The beam spread of floodlights may range from relatively narrow (10 degrees) to wide (more than 100 degrees). 167

floodlighting (illuminating engineering). A system designed for lighting a scene or object to a brightness greater than its surroundings. It may be for utility, advertising, or decorative purposes. 167

flood-lubricated bearing (rotating machinery). A bearing in which a continuous flow of lubricant is poured over the top of the bearing or journal at about normal atmospheric pressure. *See:* bearing. 63

flood projection (facsimile). The optical method of scanning in which the subject copy is floodlighted and the scanning spot is defined in the path of the reflected or transmitted light. *See:* scanning (facsimile). 12

floor acceleration (seismic qualification of Class 1E equipment for nuclear power generating stations). The acceleration of a particular building floor (or equipment mounting) resulting from the motion of a given earthquake. The maximum floor acceleration is the zero period acceleration (ZPA) of the floor response spectrum. 581

floor bushing. A bushing intended primarily to be operated entirely indoors in a substantially vertical position to carry a circuit through a floor or horizontal grounded barrier. Both ends must be suitable for operating in air. *See:* bushing. 348

floor cavity ratio (FCR) (illuminating engineering). For a cavity formed by the work-plane, the floor, and the wall surfaces between these two planes, the FCR is computed by using the distance from the floor to the work plane (h_f) as the cavity height in the equations given in the definition for cavity ratio . 167

floor lamp (illuminating engineering). A portable luminaire on a high stand suitable for standing on the floor. 167

floor trap (burglar-alarm system). A device designed to indicate an alarm condition in an electric protective circuit whenever an intruder breaks or moves a thread or conductor extending across a floor space. *See:* protective signaling. 328

flotation fluid (gyro, accelerometer) (inertial sensor). The fluid that suspends the float inside the instrument case. The float may be fully or partially floated within the fluid. The degree of flotation varies with temperature because the specific gravity of the fluid varies with temperature. In addition, the fluid provides damping. *See:* damping fluid. 46

flowchart (software). A graphical representation of the definition, analysis, or solution of a problem, in which symbols are used to represent operations, data, flow, and equipment. *See:* block diagram; data. 434

flow diagram (electronic computers). Graphic representation of a program or a routine. 210

flow of control (software). The sequence of operations performed in the execution of an algorithm. *See:* algorithm; execution. 434

flow relay (power switchgear). A relay that responds to a rate of fluid flow. 103

flow soldering. *See:* dip soldering.

flow switch (80) (power system device function numbers). A switch which operates on given values, or on a given rate of change, of flow. 402

fluctuating power (rotating machinery). A phasor quantity of which the vector represents the alternating part of the power, and that rotates at a speed equal to double the angular velocity of the current. *See:* asynchronous machine. 63

fluctuating target. A radar target whose echo amplitude varies as a function of time. *See:* target fluctuation. 13

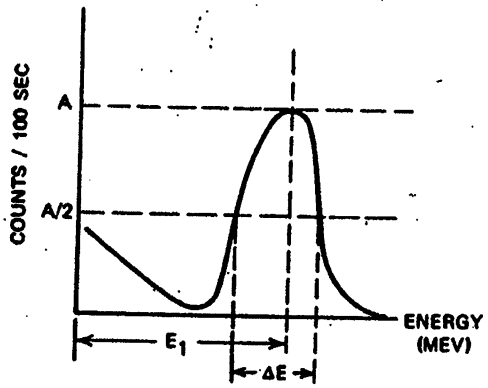
fluctuation (1) (pulse terms). Dispersion of the pulse amplitude or other magnitude parameter of the pulse waveforms in a pulse train with respect to a reference pulse amplitude or a reference magnitude. Unless otherwise specified by a mathematical adjective, peak-to-peak fluctuation is assumed. *See:* mathematical adjectives. 254

(2) (radar). *See:* target fluctuation. 13

fluctuation loss (radar). The apparent loss in radar detectability or measurement accuracy for a target of given average echo return power due to target fluctuation. It may be measured as the increase in required average echo return power of a fluctuating target as compared to a target of constant echo return, to achieve the same detectability or measuement accuracy. 13

fluctuation noise. *See:* random noise.

fluence (solar cells). The total time-integrated number of particles that cross a plane unit area from either side. 113



Full width at half maximum (in this case, Δ E).

ence to a typical distribution curve, shown in the figure, of the measurement of the energy of the gamma rays from Cs^{137} with a scintillation counter spectrometer. The measurement is made by determining the number of gamma-ray photons detected in a prescribed interval of time, having measured energies falling within a fixed energy interval (channel width) about the values of energy (channel position) taken as argument of the distribution function. The abscissa of the curve shown is energy in megaelectronvolts (MeV) units and the ordinate is counts per given time interval per megaelectronvolt energy interval. The maximum of the distribution curve shown has an energy E_1 megaelectronvolts. The height of the peak is A_1 counts/100 seconds/megaelectronvolts. The full width at half maximum ΔE is measured at a value of the ordinate equal to $A_1/2$. The percentage full width at half maximum is $100\,\Delta E/E_1$. It is an indication of the width of the distribution curve, and where (as in the example cited) the gamma-ray photons are monoenergetic, it is a measure of the resolution of the detecting instrument. When the distribution curve is a Gaussian curve, the percentage full width at half maximum is related to the standard deviation σ by

$$100\,\frac{\Delta E}{E_1} = 100 \times 2(2 \ln 2)^{1/2} \times \sigma.$$

See: **scintillation counter.** 117

(4) (sodium iodide detector). The full width of a gamma-ray peak distribution measured at half the maximum ordinate above the continuum. 423

full width at tenth maximum (FWTM)(X-ray energy spectrometers). Same as full width at half maximum (FWHM) except measurement is made at one tenth of the maximum ordinate rather than one half. 471

fully connected network (data communication). A network in which each node is directly connected with every other node. 12

fume-resistant (industrial control). So constructed that it will not be injured readily by exposure to the specified fume. 103, 202, 206

function (1)(microprocessor operating systems). A primitive operation on system-controlled resources. IEEE Std 855 (Trial Use) defines a collection of functions together with suitable input and output parameters. 478

(2) (test, measurement and diagnostic equipment). The action or purpose which a specific item is intended to perform or serve. 54

(3) (software). (A) A specific purpose of an entity or its characteristic action. (B) A subprogram that is invoked during the evaluation of an expression in which its name appears and that returns a value to the point of invocation. *See:* **subprogram; subroutine.** 434

functional adjectives (pulse terms) (1) linear. Pertaining to a feature whose magnitude varies as a function of time in accordance with the following relation or its equivalent:

$$m = a + bt$$

254

(2) exponential. Pertaining to a feature whose magnitude varies as a function of time in accordance with either of the following relations or their equivalents:

$$m = ae^{-bt}$$

$$m = a\,(1 - e^{-bt})$$

254

(3) Gaussian. Pertaining to a waveform or feature whose magnitude varies as a function of time in accordance with the following relation or its equivalent:

$$m = ae^{-b(t-c)^2}, \quad b > 0$$

254

(4) trigonometric. Pertaining to a waveform or feature whose magnitude varies as a function of time in accordance with a specified trigonometric function or by a specified relationship based on trigonometric functions (for example, cosine squared). 254

functional area(s) (nuclear power generating station). Location(s) designated within the control room to which displays and controls relating to specific function(s) are assigned. 358

functional component (power switchgear). A device which performs a necessary function for the proper operation and application of a unit of equipment. 103

functional decomposition (software). A method of designing a system by breaking it down into its components in such a way that the components correspond directly to system functions and subfunctions. *See:* **components; functions; hierarchical decomposition; system.** 434

functional design (software). The specification of the working relationships among the parts of a data processing system. *See:* **data processing system; preliminary design; specification.** 434

functional designation (abbreviation) (1) (general).

Letters, numbers, words, or combinations thereof, used to indicate the function of an item or a circuit, or of the position or state of a control of adjustment. Compare with: letter combination, reference designation, symbol for a quantity. *See:* abbreviation. 173

(2) (electric and electronics parts and equipments). Words, abbreviations, or meaningful number or letter combinations, usually derived from the function of an item (for example: slew, yaw), used on drawings, instructional material, and equipment to identify an item in terms of its function. *Note:* A functional designation is not a reference designation nor a substitute for it. 17

functional diagram (test, measurement and diagnostic equipment). A diagram that represents the functional relationships among the parts of a system. 54

functional nomenclature (generating stations electric power system). Words or terms which define the purpose, equipment, or system for which the component is required. 381

functional requirement (software). A requirement that specifies a function that a system or system component must be capable of performing. *See:* component; function; requirement; system. 434

functional specification (software). A specification that defines the functions that a system or system component must perform. *See:* component; function; performance specification; specification; system. 434

functional test (1)(ATLAS). A sequence of tests applied to a unit under test (UUT) to establish whether it is functioning correctly. 400

(2)(evaluation of thermal capability)(thermal classification of electric equipment and electrical insulation). A means of evaluation in which an insulating material, insulation system, or electric equipment is exposed to factors of influence, which simulate or are characteristic of actual service conditions. 506

(3)(test pattern language). A test in which the cells of a memory are accessed in a specific order and at a specific rate, while data is being written into them, or read from them. 463

functional test pattern. *See:* pattern.

functional unit (1). A system element that performs a task required for the successful operation of the system. *See:* system. 209

(2) (software). An entity of hardware, software, or both capable of accomplishing a specified purpose. *See:* hardware; software. 434

function check (station control and data acquisition)-(supervisory control, data acquisition, and automatic control). A check of master and remote station equipment by exercising a predefined component or capability, (1) Analog. Monitor a reference quantity (2) Control. Control and indication from a control-check relay (3) Scan. Accomplished when control function check has been performed with all remotes (4) Poll. Accomplished when analog function is performed with all remotes (5) Logging. Accomplished when results of the control function check are logged. 403, 570

function Class-A (back-up) current-limiting fuse. A fuse capable of interrupting all currents from the rated maximum interrupting current down to the rated minimum interrupting current. *Note:* The rated minimum interrupting current for such fuses is higher than the minimum melting current that causes melting of the fusible element in one hour. 103

function Class-G (general purpose) current-limiting fuse (as applied to a high-voltage current-limiting fuse). A fuse capable of interrupting all currents from the rated maximum interrupting current down to the current that causes melting of the fusible element in one hour. 103

function code (f) (subroutines for CAMAC). The symbol *f* represents an integer which is the function code for a CAMAC action. 410

function codes (fa) (subroutines for CAMAC). The symbol *fa* represents an array of integers, each of which is the function code for a CAMAC action. The length of *fa* is given by the value of the first element of *cb* at the time the subroutine is executed. *See:* control block. 410

function, coupling (control systems). A mathematical, graphical, or tabular statement of the influence which one element or subsystem has on another element or subsystem, expressed as the effect.cause ratio of related variables or their transforms. *Note:* For a multiterminal system described by m differential equations and having m input transforms $R_1 \ldots R_m$ and m output transforms $C_1 \ldots C_m$, the coupling functions consist of all effect.cause ratios which can be formed from transforms bearing unlike-numbered subscripts. 56

function, describing (nonlinear element under periodic input) (control system, feedback). A transfer function based solely on the fundamental, ignoring other frequencies. *Note:* This equivalent linearization implies amplitude dependence with or without frequency dependence. *See:* control system, feedback. 56, 329

function, error transfer (closed loop) (control system, feedback). The transfer function obtained by taking the ratio of the Laplace transform of the error signal to the Laplace transform of its corresponding input signal. *See:* control system, feedback. 56, 329

function generator (1) (analog computer). A computing element whose output is a specified nonlinear function of its input or inputs. Normal usage excludes multipliers and resolvers. 9

(2) (electric power systems). A device in which a mathematical function such as y, $f(x)$ can be stored so that for any input equal to x, an output equal to $f(x)$ will be obtained. *See:* speed-governing system. 94

function generator, bivariant. A function generator having two input variables. *See:* electronic analog computer. 9

function generator, card set (analog computer). A diode function generator whose values are stored and set by means of a punched card and mechanical card reading device. 9

function generator, curve-follower (analog computer).

U.S. EPA Hearing

Case 5:03-cv-04669-JW Document 238-3 Filed 08/27/2004 Page 8 of 8

required outputs (software verification and validation plans). The set of items produced as a result of performing the minimum verification and validation (V&V) tasks mandated within any life-cycle phase. 511

required reserve (power operations). The system planned reserve capability needed to ensure a specified standard of service. 516

required response spectrum (RRS) (1) (seismic qualification of Class 1E equipment for nuclear power generating stations). The response spectrum issued by the user or his agent as part of his specifications for qualification or artificially created to cover future applications. The RRS constitutes a requirement to be met. 581

(2) (valve actuators). The required response spectrum issued by the user or his agent as part of his specifications for proof testing, or artificially created to cover future applications. The RRS constitutes a requirement to be met. 492

(3) (nuclear power generating stations) (seismic qualification of class 1E equipment). The response spectrum issued by the user or his agent as part of his specifications for proof testing, or artificially created to cover future applications. The RRS constitutes a requirement to be met. 28

(4) (seismic testing of relays). The response spectrum issued by the user or his agent as part of his specifications for proof testing, or artificially created to cover future applications. The RRS constitutes a requirement to be met. 392

required time (availability). The period of time during which the user requires the item to be in a condition to perform its required function. 164

requirement (software). (1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. The set of all requirements forms the basis for subsequent development of the system or system component. *See:* **component; document; requirements analysis; requirements phase; requirements specification; specification; system.** 434

requirements analysis (software). (1) The process of studying user needs to arrive at a definition of system or software requirements. (2) The verification of system or software requirements. *See:* **software requirement; system; verification.** 434

requirements inspection. *See:* **inspection.**

requirements phase (1)(software). The period of time in the software life cycle during which the requirements for a software product, such as the functional and performance capabilities, are defined and documented. 434

(2)(software verification and validation plans). The period of time in the software life cycle during which the requirements, such as functional and performance capabilities for a software product, are defined and documented. 511

requirements specification (software). A specification that sets forth the requirements for a system or system component, for example, a software configuration item. Typically included are functional requirements, performance requirements, interface requirements, design requirements, and development standards. 434

requirements specification language (software). A formal language with special constructs and verification protocols used to specify, verify, and document requirements. *See:* **document requirement; formal language; verification.** 434

requirements verification. *See:* **verification.**

reradiation. (1) The scattering of incident radiation, or (2) the radiation of signals amplified in a radio receiver. *See:* **radio receiver.** 328

rerecording (electroacoustics). The process of making a recording by reproducing a recorded sound source and recording this reproduction. *See:* **dubbing.** 176

rerecording system (electroacoustics). An association of reproducers, mixers, amplifiers, and recorders capable of being used for combining or modifying various sound recordings to provide a final sound record. *Note:* Recording of speech, music, and sound effects may be so combined. *See:* **dubbing; phonograph pickup.** 176

re-refining (insulating oil). The use of primary refining processes on used electrical insulating liquids that are suitable for further use as electrical insulating liquids. *Note:* Techniques may include a combination of distillation and acid, clay or hydrogen treating, and other physical and chemical means. 461

rering signal (telephone switching systems). A signal initiated by an operator at the calling end of an established connection to recall the operator at the called end or the customer at either end. 55

rerun point (computing systems). The location in the sequence of instructions in a computer program at which all information pertinent to the rerunning of the program is available. 255, 77, 54

reseal voltage rating (surge arrester). The maximum arrester recovery voltage permitted for a specified time following one or more unit operation(s) with discharge currents of specified magnitude and duration. 62

reserve (1) (test, measurement and diagnostic equipment). The setting aside of a specific portion of memory for a storage area. 54

(2) (electric power system) (generating stations electric power system). A qualifying term used to identify equipment and capability that is available and is in excess of that required for the load. *Note:* The reserve may be connected to the system and partially loaded or may be made available by closing switches, contactors, or circuit breakers. Reserve not in operation and requiring switching is sometimes called standby equipment. 381

(3) (power operations). *See:* **customer generation reserve; electrical reserve; installed reserve; interruptible load reserve; nonspinning reserve; operating re-**

EXHIBIT 4
(Part 1 of 2)

S

Ninth New Collegiate Dictionary

A Merriam-Webster®

MERRIAM-WEBSTER INC., *Publishers*
Springfield, Massachusetts, U.S.A.

RCL011389

A GENUINE MERRIAM-WEBSTER

The name *Webster* alone is no guarantee of excellence. It is used by a number of publishers and may serve mainly to mislead an unwary buyer.

A Merriam-Webster® is the registered trademark you should look for when you consider the purchase of dictionaries or other fine reference books. It carries the reputation of a company that has been publishing since 1831 and is your assurance of quality and authority.

Copyright © 1987 by Merriam-Webster Inc.

Philippines Copyright 1987 by Merriam-Webster Inc.

Library of Congress Cataloging in Publication Data
Main entry under title:

Webster's ninth new collegiate dictionary.

Based on Webster's third new international dictionary.
Includes index.
1. English language—Dictionaries. I. Merriam-Webster Inc.
PE1628.W5638 1987 423 86-23801
ISBN 0-87779-508-8
ISBN 0-87779-509-6 (indexed)
ISBN 0-87779-510-X (deluxe)

Webster's Ninth New Collegiate Dictionary principal copyright 1983

COLLEGIATE trademark Reg. U.S. Pat. Off.

All rights reserved. No part of this book covered by the copyrights hereon may be reproduced or copied in any form or by any means—graphic, electronic, or mechanical, including photocopying, taping, or information storage and retrieval systems—without written permission of the publisher.

Made in the United States of America

2526RMcN87

Case 5:03-cv-04669-JW Document 238-4 Filed 08/27/2004 Page 4 of 11

ac·tion \'ak-shən\ *n* (14c) **1** : a proceeding in a court of justice by which one demands or enforces one's right **2** : the bringing about of an alteration by force or through a natural agency **3** : the manner or method of performing: **a** : the deportment of an actor or speaker or his expression by means of attitude, voice, and gesture **b** : the style of movement of the feet and legs (as of a horse) **c** : a function of the body or one of its parts **4** : an act of will **5 a** : a thing done : DEED **b** : the accomplishment of a thing usu. over a period of time, in stages, or with the possibility of repetition ⟨an ~, the product and expression of exerted force —Thomas Carlyle⟩ **c** *pl* : BEHAVIOR, CONDUCT ⟨unscrupulous ~s⟩ **d** : INITIATIVE, ENTERPRISE ⟨a man of ~⟩ **6 a** (1) : an engagement between troops or ships (2) : combat in war ⟨gallantry in ~⟩ **b** (1) : an event or series of events forming a literary composition (2) : the unfolding of the events of a drama or work of fiction : PLOT (3) : the movement of incidents in a plot **c** : the combination of circumstances that constitute the subject matter of a painting or sculpture **7 a** : an operating mechanism **b** : the manner in which a mechanism operates **8 a** : the price movement and trading volume of a commodity, security, or market **b** : the process of betting including the offering and acceptance of a bet and determination of a winner **9** : the most vigorous, productive, or exciting activity in a particular field, area, or group ⟨they itch to go where the ~ is —D. J. Henahan⟩

ac·tion·able \'ak-sh(ə-)nə-bəl\ *adj* (1591) : subject to or affording ground for an action or suit at law — **ac·tion·ably** \-blē\ *adv*

ac·tion·less \'ak-shən-ləs\ *adj* (1817) : marked by inaction : IMMOBILE

action painting *n* (1952) : abstract expressionism marked esp. by the use of spontaneous techniques (as dribbling, splattering, or smearing) — **action painter** *n*

action potential *n* (1926) : a momentary change in electrical potential (as between the inside of a nerve cell and the extracellular medium) that occurs when a cell or tissue has been activated by a stimulus

ac·ti·vate \'ak-tə-ˌvāt\ *vb* **-vat·ed; -vat·ing** *vt* (1626) : to make active or more active: as **a** (1) : to make (as molecules) reactive or more reactive (2) : to convert (as a provitamin) into a biologically active derivative **b** : to make (a substance) radioactive, luminescent, photosensitive, or photoconductive **c** : to treat (as carbon or alumina) so as to improve adsorptive properties **d** : to aerate (sewage) so as to favor the growth of organisms that decompose organic matter **e** (1) : to set up or formally institute (as a military unit) with the necessary personnel and equipment (2) : to put (an individual or unit) on active duty ~ *vi* : to become active — **ac·ti·va·tion** \ˌak-tə-'vā-shən\ *n* — **ac·ti·va·tor** \'ak-tə-ˌvāt-ər\ *n*

activated carbon *n* (1921) : a highly adsorbent powdered or granular carbon made usu. by carbonization and chemical activation and used chiefly for purifying by adsorption — called also *activated charcoal*

activation analysis *n* (ca. 1949) : analysis to determine chemical elements in a material by bombarding it with neutrons to produce radioactive atoms whose radiations are characteristic of the elements present

activation energy *n* (1940) : the minimum amount of energy required to convert a normal stable molecule into a reactive molecule

ac·tive \'ak-tiv\ *adj* [ME, fr. MF or L; MF *actif*, fr. L *activus*, fr. *actus*, pp. of *agere* to drive, do — more at AGENT] (14c) **1** : characterized by action rather than by contemplation or speculation **2** : productive of action or movement **3 a** *of a verb form or voice* : asserting that the person or thing represented by the grammatical subject performs the action represented by the verb *hits* in "he hits the ball" is ~ **b** : expressing action as distinct from mere existence or state **4** : quick in physical movement : LIVELY **5** : marked by vigorous activity : BUSY ⟨the stock market was ~⟩ **6** : requiring vigorous action or exertion ⟨~ sports⟩ **7** : having practical operation or results : EFFECTIVE ⟨an ~ law⟩ **8 a** : disposed to action : ENERGETIC ⟨~ interest⟩ **b** : engaged in an action or activity ⟨an ~ club member⟩ **9** : engaged in full-time service esp. in the armed forces ⟨~ duty⟩ **10** : marked by present operation, transaction, movement, or use ⟨~ account⟩ **11 a** : capable of acting or reacting : ACTIVATED ⟨~ nitrogen⟩ ⟨~ charcoal⟩ **b** : tending to progress or to cause degeneration ⟨~ tuberculosis⟩ **c** : exhibiting optical activity **d** *of an electronic element* : capable of controlling voltages or currents **e** : requiring the expenditure of energy ⟨~ calcium ion uptake⟩ **12** : still eligible to win the pot in poker **13** : moving down the line : visiting in the set — used of couples in contradances or square dances — **active** *n* — **ac·tive·ly** *adv* — **ac·tive·ness** *n*

active immunity *n* (1911) : usu. long-lasting immunity that is acquired through production of antibodies within the organism in response to the presence of antigens — compare PASSIVE IMMUNITY

active transport *n* (ca. 1963) : movement of a chemical substance by the expenditure of energy through a gradient (as across a cell membrane) in concentration or electrical potential and opposite to the direction of normal diffusion

ac·tiv·ism \'ak-tə-ˌviz-əm\ *n* (1915) : a doctrine or practice that emphasizes direct vigorous action (as a mass demonstration) in support of or opposition to one side of a controversial issue — **ac·tiv·ist** \-vəst\ *n or adj* — **ac·tiv·is·tic** \ˌak-tə-'vis-tik\ *adj*

ac·tiv·i·ty \ak-'tiv-ət-ē\ *n, pl* **-ties** (15c) **1** : the quality or state of being active **2** : vigorous or energetic action : LIVELINESS **3** : natural or normal function: as **a** : a process (as digestion) that an organism carries on or participates in by virtue of being alive **b** : a similar process actually or potentially involving mental function; *specif* : an educational procedure designed to stimulate learning by firsthand experience **4** : an active force **5 a** : a pursuit in which a person is active **b** : a form of organized, supervised, often extracurricular recreation **6** : an organizational unit for performing a specific function; *also* : its function or duties

act of God (1859) : an extraordinary interruption by a natural cause (as a flood or earthquake) of the usual course of events that experience, prescience, or care cannot reasonably foresee or prevent

ac·to·my·o·sin \ˌak-tə-'mī-ə-sən\ *n* [ISV *actin* + *-o-* + *myosin*] (1942) : a viscous contractile complex of actin and myosin concerned together with ATP in muscular contraction

ac·tor \'ak-tər *also* -ˌtȯ(ə)r\ *n* (15c) **1** : one that acts : DOER **2 a** : one who represents a character in a dramatic production **b** : a theatrical performer **c** : one that behaves as if acting a part **3** : one that takes part in any affair — **ac·tor·ish** \-tə-rish\ *adj*

act out *vt* (1611) **1 a** : to represent in action ⟨children *act out* what they read⟩ **b** : to translate into action ⟨unwilling to *act out* their beliefs⟩ **2** : to express (as an impulse or a fantasy) directly in overt behavior without modification to comply with social norms

ac·tress \'ak-trəs\ *n* (1676) : a woman who is an actor

Acts \'akts\ *n pl but sing in constr* : a book in the New Testament narrating the beginnings of the Christian Church — called also *Acts of the Apostles*; see BIBLE table

ac·tu·al \'ak-ch(ə-w)əl, -shə-w)əl\ *adj* [ME *actuel*, fr. MF, fr. LL *actualis*, fr. L *actus* act] (14c) **1** *obs* : ACTIVE **2 a** : existing in act and not merely potentially **b** : existing in fact or reality ⟨~ and imagined conditions⟩ **c** : not false or apparent ⟨~ costs⟩ **3** : existing or occurring at the time : CURRENT ⟨caught in the ~ commission of a crime⟩

actual cash value *n* (1946) : money equal to the cost of replacing lost, stolen, or damaged property after depreciation

ac·tu·al·i·ty \ˌak-chə-'wal-ət-ē, ak-shə-\ *n, pl* **-ties** (1652) **1** : the quality or state of being actual **2** : something that is actual : FACT, REALITY ⟨possible risks which have been seized upon as *actualities* —T. S. Eliot⟩

ac·tu·al·ize \'ak-ch(ə-w)ə-ˌlīz, -sh(ə-w)ə-ˌlīz\ *vb* **-ized; -iz·ing** *vt* (1701) : to make actual ~ *vi* : to become actual — **ac·tu·al·iz·a·tion** \ˌak-ch(ə-w)ə-lə-'zā-shən, -sh(ə-w)ə-lə-\ *n*

ac·tu·al·ly *adv* \'ak-ch(ə-w)ə-lē, -sh(ə-w)ə-lē, 'aks(h)-lē\ (15c) **1** : in act or in fact : REALLY ⟨nominally but not ~ independent —Karl Loewenstein⟩ **2** : in point of fact : in truth ⟨he ~ spoke Latin⟩

ac·tu·ar·i·al \ˌak-chə-'wer-ē-əl, -shə-\ *adj* (1869) **1** : of or relating to actuaries **2** : relating to statistical calculation esp. of life expectancy — **ac·tu·ar·i·al·ly** \-ē-ə-lē\ *adv*

ac·tu·ary \'ak-chə-ˌwer-ē, -shə-\ *n, pl* **-ar·ies** [L *actuarius* shorthand writer, fr. *actum* record — more at ACT] (1553) **1** *obs* : CLERK, REGISTRAR **2** : one who calculates insurance and annuity premiums, reserves, and dividends

ac·tu·ate \'ak-chə-ˌwāt, -shə-\ *vt* **-at·ed; -at·ing** [ML *actuatus*, pp. of *actuare*, to execute, fr. L *actus* act] (1645) **1** : to put into mechanical action or motion **2** : to move to action *syn* see MOVE — **ac·tu·a·tion** \ˌak-chə-'wā-shən, -shə-\ *n*

ac·tu·a·tor \'ak-chə-ˌwāt-ər, -shə-\ *n* (ca. 1864) : one that actuates; *specif* : a mechanism for moving or controlling something indirectly instead of by hand

act up *vi* (1903) **1** : to act in a way different from that which is normal or expected: as **a** : to behave in an unruly, recalcitrant, or capricious manner **b** : SHOW OFF **c** : to function improperly ⟨this typewriter is *acting up* again⟩ **2** : to become active or acute after being quiescent ⟨her rheumatism started to *act up*⟩

acu·ity \ə-'kyü-ət-ē, a-\ *n, pl* **-ities** [MF *acuité*, fr. OF *agüeté*, fr. *agu* sharp, fr. L *acutus*] (1543) : keenness of perception : SHARPNESS

acu·le·ate \ə-'kyü-lē-ət\ *adj* [L *aculeatus* having stings, fr. *aculeus*, dim. of *acus*] (1661) : having a sting ⟨~ insects⟩

acu·men \ə-'kyü-mən, 'ak-yə-mən\ *n* [L *acumin-*, *acumen*, lit., point, fr. *acuere*] (1531) : keenness and depth of perception, discernment, or discrimination esp. in practical matters : SHREWDNESS *syn* see DISCERNMENT

acu·mi·nate \ə-'kyü-mə-nət\ *adj* (1646) : tapering to a slender point

acu·pres·sure \'ak-(y)ə-ˌpresh-ər\ *n* (1859) : SHIATSU

acu·punc·ture \-ˌpəŋ(k)-chər\ *n* [L *acus* + E *puncture*] (ca. 1860) : an orig. Chinese practice of puncturing the body (as with needles) at specific points to cure disease or relieve pain (as in surgery) — **acu·punc·tur·ist** \-ˌpəŋ(k)-chə-rəst\ *n*

acute \ə-'kyüt\ *adj* **acut·er; acut·est** [L *acutus*, pp. of *acuere* to sharpen, fr. *acus* needle; akin to L *acer* sharp — more at EDGE] (14c) **1 a** (1) : characterized by sharpness or severity ⟨~ pain⟩ ⟨an ~ infection⟩ (2) : having a sudden onset, sharp rise, and short course ⟨~ disease⟩ **b** : lasting a short time ⟨~ experiments⟩ **2** : ending in a sharp point: as **a** : being or forming an angle measuring less than 90 degrees ⟨~ angle⟩ **b** : composed of acute angles ⟨~ triangle⟩ **3 a** : marked by keen discernment or intellectual perception esp. of subtle distinctions : PENETRATING ⟨an ~ thinker⟩ **b** : responsive to slight impressions or stimuli ⟨~ observer⟩ **4** : felt, perceived, or experienced intensely ⟨~ distress⟩ **5** : seriously demanding urgent attention ⟨an ~ housing shortage⟩ **6 a** *of an accent mark* : having the form **b** : marked with an acute accent **c** : of the variety indicated by an acute accent — **acute·ly** *adv* — **acute·ness** *n*

syn ACUTE, CRITICAL, CRUCIAL mean of uncertain outcome. ACUTE stresses intensification of conditions leading to a culmination or breaking point; CRITICAL adds to ACUTE implications of imminent change, of attendant suspense, and of decisiveness in the outcome; CRUCIAL suggests a dividing of the ways and often a test or trial involving the determination of a future course or direction. *syn* see in addition SHARP

acy·clic \(ˌ)ā-'sī-klik, -'sik-lik\ *adj* (1878) **1** : not cyclic; *esp* : not disposed in cycles or whorls **2** : having an open-chain structure; *esp* : ALIPHATIC ⟨an ~ compound⟩

acyl \'as-əl\ *n* [ISV, fr. *acid*] (1901) : a radical derived usu. from an organic acid by removal of the hydroxyl from all acid groups

1ad \'ad\ *n, often attrib* (1841) **1** : ADVERTISEMENT **2** : ADVERTISING

2ad *n* (1947) : ADVANTAGE 4

ad- *or* **ac-** *or* **af-** *or* **ag-** *or* **al-** *or* **ap-** *or* **as-** *or* **at-** *prefix* [ME, fr. MF, OF, & L; MF, fr. OF, fr. L, fr. *ad* — more at AT] **1** : to : toward, — usu. *ac-* before *c*, *k*, or *q* ⟨*ac*culturation⟩ and *af-* before *f* and *ag-* before *g* ⟨*ag*grade⟩ and *al-* before *l* ⟨*al*literation⟩ and *ap-* before *p* ⟨*ap*proximal⟩ and *as-* before *s* ⟨*as*sussive⟩ and *at-* before *t* ⟨*at*tune⟩ and *ad-* before other sounds but sometimes *ad-* even before one of the listed consonants ⟨*ad*sorb⟩ **2** : near : adjacent to — in this sense always in the form *ad-* ⟨*ad*renal⟩

-ad \ˌad, əd\ *adv suffix* [L *ad*] : in the direction of : toward ⟨cephal*ad*⟩

ad·age \'ad-ij\ *n* [MF, fr. L *adagium*, fr. *ad-* + *-agium* (akin to *aio* I say)] (1548) : a saying often in metaphorical form that embodies a common observation

1ada·gio \ə-'däj-(ˌ)ō, ä-, -'däzh-\ *adv or adj* [It, fr. *ad* to + *agio* ease, fr. LL *adjacens* near at hand — more at EASE] (1724) : in an easy graceful manner : SLOWLY — used chiefly as a direction in music

2adagio *n, pl* **-gios** (1754) **1** : a musical composition or movement in adagio tempo **2** : a ballet duet by a man and woman or a mixed trio displaying difficult feats of balance, lifting, or spinning

RCL011391

Case 5:03-cv-04669-JW Document 238-4 Filed 08/27/2004 Page 5 of 11

Ar·e·op·a·gus \-ˈäp-ə-gəs\ *n* [L, fr. Gk *Areios pagos*, fr. *Areios pagos* (lit., hill of Ares), a hill in Athens where the tribunal met] (1586) : the supreme tribunal of Athens

Ar·es \ˈa(ə)r-(ˌ)ēz, ˈe(ə)r-\ *n* [Gk *Arēs*] : the Greek god of war — compare MARS

arête \ə-ˈrāt\ *n* [F, lit., fish bone, fr. LL *arista*, fr. L, beard of grain] (1838) : a sharp-crested ridge in rugged mountains

Ar·e·thu·sa \ˌar-ə-ˈthy(y)ü-zə\ *n* [L, fr. Gk *Arethousa*] : a wood nymph who is changed into a spring while fleeing the advances of the river-god Alpheus

ar·gali \ˈär-gə-lē\ *n* [Mongolian] (1774) : a large wild sheep (*Ovis ammon*) of Asia that is noted for its large horns; *also* : any of several other large wild sheep (as the bighorn)

Ar·gand diagram \ˌär-gän-, -ˌgan-\ *n* [John Robert *Argand* †1825 Fr. mathematician] (1908) : a conventional diagram in which the complex number $x + iy$ is represented by the point whose rectangular coordinates are x and y

ar·gent \ˈär-jənt\ *n* [ME, fr. MF & L; MF, fr. L *argentum*, akin to L *arguere* to make clear, Gk *argyros* silver, *argos* white] (15c) **1** *archaic* : the metal silver; *also* : WHITENESS **2** : the heraldic color silver or white — **argent** *adj*

ar·gen·tic \ärˈjent-ik\ *adj* (1868) : of, relating to, or containing silver esp. when bivalent

ar·gen·tif·er·ous \ˌär-jənˈtif-(ə-)rəs\ *adj* (1801) : producing or containing silver

¹ar·gen·tine \ˈär-jən-ˌtīn, -ˌtēn\ *adj* (15c) : SILVER, SILVERY

²argentine *n* (1577) : SILVER; *also* : any of various materials resembling it

ar·gen·tite \ˈär-jən-ˌtīt\ *n* (1837) : native silver sulfide Ag_2S having a metallic luster and dark lead-gray color and constituting a valuable ore of silver

ar·gen·tous \ärˈjent-əs\ *adj* (1869) : of, relating to, or containing silver esp. when univalent

ar·gil \ˈär-jəl\ *n* [ME, fr. L *argilla*, fr. Gk *argillos*; akin to Gk *argos* white] (14c) : CLAY; *esp* : POTTER'S CLAY

ar·gil·la·ceous \ˌär-jə-ˈlā-shəs\ *adj* (ca. 1731) : of, relating to, or containing clay or clay minerals : CLAYEY

ar·gil·lite \ˈär-jə-ˌlīt\ *n* (1795) : a compact argillaceous rock differing from shale in being cemented by silica and from slate in having no slaty cleavage

ar·gi·nase \ˈär-jə-ˌnās, -ˌnāz\ *n* [ISV] (1904) : a crystalline enzyme that converts naturally occurring arginine into ornithine and urea

ar·gi·nine \ˈär-jə-ˌnēn\ *n* [G *arginin*] (1886) : a crystalline basic amino acid $C_6H_{14}N_4O_2$ derived from guanidine

Ar·give \ˈär-ˌjīv, -ˌgīv\ *adj* [L *Argivus*, fr. Gk *Argeios*, lit., of Argos, fr. *Argos* city-state of ancient Greece] (1598) : of or relating to the Greeks or Greece and esp. the Achaean city of Argos or the surrounding territory of Argolis — **Argive** *n*

ar·gle–bar·gle \ˌär-gəl-ˈbär-gəl\ *n* [redupl. of Sc & E *argle*, alter. of *argue*] *chiefly Brit* (1872) : ARGY-BARGY

Ar·go \ˈär-(ˌ)gō\ *n* [L (gen. *Argus*), fr. Gk *Argō*] : a large constellation in the southern hemisphere lying principally between Canis Major and the Southern Cross

ar·gol \ˈär-ˌgōl\ *n* [ME *argoille*, fr. AF *argoil*] (14c) : crude tartar deposited in wine casks during aging

ar·gon \ˈär-ˌgän\ *n* [Gk, neut. of *argos* idle, lazy, fr. *a-* + *ergon* work; fr. its relative inertness — more at WORK] (ca. 1890) : a colorless odorless inert gaseous element found in the air and in volcanic gases and used esp. as a filler for electric bulbs — see ELEMENT table

ar·go·naut \ˈär-gə-ˌnȯt, -ˌnät\ *n* [L *Argonautes*, fr. Gk *Argonautēs*, fr. *Argō*, ship in which the Argonauts sailed + *nautēs* sailor — more at NAUTICAL] (14c) **1** *cap* : any of a band of heroes sailing with Jason in quest of the Golden Fleece **b** : an adventurer engaged in a quest **2** : PAPER NAUTILUS

ar·go·sy \ˈär-gə-sē\ *n, pl* **-sies** [modif. of It *ragusea* Ragusan vessel, fr. *Ragusa*, Dalmatia (now Dubrovnik, Yugoslavia)] (1577) **1** : a large ship; *esp* : a large merchant ship (three of your *argosies* are . . . come to harbor—Shak.) **2** : a fleet of ships **3** : a rich supply (an ~ of railway folklore —F.P. Donovan)

ar·got \ˈär-gət, -(ˌ)gō\ *n* [F] (1860) : an often more or less secret vocabulary and idiom peculiar to a particular group (the American Negro has . . . developed his own ~, partly to put the white man off, partly to put him down —Daniel Stern)

ar·gu·able \ˈär-gyə-wə-bəl\ *adj* (ca. 1611) **1** : open to argument, dispute, or question **2** : that can be plausibly or convincingly argued — **ar·gu·ably** \-blē\ *adv*

ar·gue \ˈär-(ˌ)gyü, -gyə(-w)\ *vb* **ar·gued; ar·gu·ing** [ME *arguen*, fr. MF *arguer* to accuse, reason & L *arguere* to make clear; MF *arguer*, fr. L *argutare* to prate, fr. *argutus* clear, noisy, fr. pp. of *arguere* — more at ARGENT] *vi* (14c) **1** : to give reasons for or against something : REASON **2** : to contend or disagree in words : DISPUTE ~ *vt* **1** : to give evidence of : INDICATE **2** : to consider the pros and cons of : DISCUSS **3** : to prove or try to prove by giving reasons : MAINTAIN **4** : to persuade by giving reasons : INDUCE *syn* see DISCUSS — **ar·gu·er** \-gyə-wər\ *n*

ar·gu·fy \ˈär-gyə-ˌfī\ *vb* **-fied; -fy·ing** *vi* (1771) : DISPUTE, DEBATE ~ *vi* : WRANGLE — **ar·gu·fi·er** \-ˌfī-(ə)r\ *n*

ar·gu·ment \ˈär-gyə-mənt\ *n* [ME, fr. MF, fr. L *argumentum*, fr. *arguere*] (14c) **1** *obs* : an outward sign : INDICATION **2** *a* : a reason given in proof or rebuttal **b** : discourse intended to persuade **3** *a* : the act or process of arguing : ARGUMENTATION **b** : a coherent series of statements leading from a premise to a conclusion **c** : QUARREL, DISAGREEMENT **4** : an abstract or summary esp. of a literary work (a later editor added an ~ *to* the poem) **5** : the subject matter esp. of a literary work **6** *a* : one of the independent variables upon whose value that of a function depends **b** : the angle that fixes the direction of a complex number (if $a + bi$ is written as $re^{i\theta} = r(\cos\theta + i\sin\theta)$ then θ is the ~)

ar·gu·men·ta·tion \ˌär-gyə-mən-ˈtā-shən, -ˌmen-\ *n* (15c) **1** : the act or process of forming reasons and of drawing conclusions and applying them to a case in discussion **2** : DEBATE, DISCUSSION

ar·gu·men·ta·tive \ˌär-gyə-ˈment-ət-iv\ *also* **ar·gu·men·tive** \-ˈment-iv\ *adj* (15c) **1** : characterized by argument : CONTROVERSIAL **2** : given to argument : DISPUTATIOUS — **ar·gu·men·ta·tive·ly** *adv*

ar·gu·men·tum \ˌär-gyə-ˈment-əm\ *n, pl* **-men·ta** \-ˈment-ə\ [L] (1697) : ARGUMENT 3b

Ar·gus \ˈär-gəs\ *n* [L, fr. Gk *Argos*] **1** : a hundred-eyed monster of Greek legend **2** : a watchful guardian

Ar·gus–eyed \ˌär-gə-ˈsīd\ *adj* (1603) : vigilantly observant

ar·gy–bar·gy \ˌär-jē-ˈbär-jē, ˌär-gē-ˈbär-gē\ *n* [redupl. of Sc & E *argy*, alter. of *argue*] *chiefly Brit* (ca. 1887) : a lively discussion : ARGUMENT, DISPUTE

ar·gyle *also* **ar·gyll** \ˈär-ˌgīl, är-\ *n, often cap* [*Argyle, Argyll*, branch of the Scottish clan of Campbell, fr. whose tartan the design was adapted] (1899) : a geometric knitting pattern of varicolored diamonds in solid and outline shapes on a single background color; *also* : a sock knit in this pattern

Ar·gy·rol \ˈär-jə-ˌrōl, -ˌrȯl\ *trademark* — used for a silver-protein compound whose aqueous solution is used as a local antiseptic esp. on mucous membranes

ar·hat \ˈär-(ˌ)hət\ *n* [Skt, fr. prp. of *arhati* he deserves; akin to Gk *phein* to gain] (1850) : a Buddhist who has reached the stage of enlightenment — **ar·hat·ship** \-ˌship\ *n*

aria \ˈär-ē-ə\ *n* [It, lit., atmospheric air, modif. of L *aer*] (ca. 1724) : AIR, MELODY, TUNE; *specif* : an accompanied elaborate melody sung in an opera) by a single voice **2** : a striking solo performance (as in movie)

Ar·i·ad·ne \ˌar-ē-ˈad-nē\ *n* [L, fr. Gk *Ariadnē*] : a daughter of Minos who helps Theseus escape from the labyrinth

¹Ar·i·an \ˈar-ē-ən, ˈer-\ *adj* (14c) : of or relating to Arius or his doctrine esp. that the Son is not of the same substance as the Father but was created as an agent for creating the world — **Ar·i·an·ism** \-ə-ˌniz-əm\

²Arian *n* (14c) : a supporter of Arian doctrines

³Ar·i·an \ˈer-ē-ən, ˈar-\ *n* (1967) : ARIES 2b

-ar·i·an \ˈer-ē-ən, ˈar-\ *n suffix* [L *-arius* -ary] **1** : believer (necessitarian) : advocate (latitudinarian) **2** : producer (disciplinarian)

ari·bo·fla·vin·osis \ˌa-ˌrī-bō-ˌflā-və-ˈnō-səs\ *n* [NL, fr. *a-* + *riboflavin* + *-osis*] (1939) : a deficiency disease due to inadequate intake of riboflavin

ar·id \ˈar-əd\ *adj* [F or L; F *aride*, fr. L *aridus* — more at ARDOR] (1662) **1** : excessively dry; *specif* : having insufficient rainfall to support agriculture **2** : lacking in interest and life : JEJUNE — **arid·i·ty** \ə-ˈrid-ət-ē\ *n* — **ar·id·ness** \ˈar-əd-nəs\ *n*

Ar·i·el \ˈar-ē-əl, ˈer-\ *n* **1** : a prankish spirit in Shakespeare's *The Tempest* **2** : the inner satellite of Uranus

Ar·i·es \ˈer-(ē-)ēz, ˈar-\ *n* [L (gen. *Arietis*), lit., ram; akin to Gk *eriphos* kid, OIr *heirp* doe] **1** : a constellation between Pisces and Taurus pictured as a ram **2** *a* : the first sign of the zodiac in astrology — see ZODIAC table **b** : one born under this sign

ari·et·ta \ˌär-ē-ˈet-ə, ˌar-\ *n* [It, dim. of *aria*] (ca. 1724) : a short aria

aright \ə-ˈrīt\ *adv* [ME, fr. OE *āriht*, fr. *ā-* + *riht* right] (bef. 12c) : RIGHTLY, CORRECTLY (if I remember ~)

ari·il \ˈar-əl\ *n* [prob. fr. NL *arillus*, fr. ML, raisin, grape seed] (1790) : an exterior covering or appendage of some seeds that develops after fertilization as an outgrowth from the ovule stalk — **ari·il·ate** \ˈar-ə-ˌlāt\ *adj*

ari·o·so \ˌär-ē-ˈō-(ˌ)sō, -(ˌ)zō\ *n, pl* **-sos** *also* **-si** \-(ˌ)sē, -(ˌ)zē\ [It, fr. *aria*] (ca. 1724) : a musical passage or composition having a mixture of the recitative and metrical song

arise \ə-ˈrīz\ *vi* **arose** \-ˈrōz\; **aris·en** \-ˈriz-ᵊn\; **aris·ing** \-ˈrī-ziŋ\ [ME *arisen*, fr. OE *ārisan*, fr. *ā-*, perfective prefix + *risan* to rise — more at ABIDE] (bef. 12c) **1** : to get up : RISE **2** *a* : to originate from a source **b** : to come into being or to attention **3** : ASCEND *syn* see SPRING

aris·ta \ə-ˈris-tə\ *n, pl* **-tae** \-(ˌ)tē, -ˌtī\ *or* **-tas** [NL, fr. L, beard of grain] (1691) : a bristlelike structure or appendage — **aris·tate** \-ˌtāt\ *adj*

aris·to \ə-ˈris-(ˌ)tō\ *n, pl* **-tos** [by shortening] *chiefly Brit* (1864) : ARISTOCRAT

ar·is·toc·ra·cy \ˌar-ə-ˈstäk-rə-sē\ *n, pl* **-cies** [MF & LL; MF *aristocratie*, fr. LL *aristocratia*, fr. Gk *aristokratia*, fr. *aristos* best + *-kratia* -cracy] (1561) **1** : government by the best individuals or by a small privileged class **2** *a* : a government in which power is vested in a minority consisting of those believed to be best qualified **b** : a state with such a government **3** : a governing body or upper class made up of an hereditary nobility **4** : the aggregate of those believed to be superior

aris·to·crat \ə-ˈris-tə-ˌkrat, a-; ˈar-ə-stə-\ *n* (1789) **1** : a member of an aristocracy; *esp* : NOBLE **2** *a* : one who has the bearing and viewpoint typical of the aristocracy **b** : one who favors aristocracy **3** : one believed to be superior of its kind (the ~ of Southern resorts —*Southern Living*)

aris·to·crat·ic \ə-ˌris-tə-ˈkrat-ik, (ˌ)a-ˌris-tə-, ˌar-ə-stə-\ *adj* [MF *aristocratique*, fr. ML *aristocraticus*, fr. Gk *aristokratikos*, fr. *aristos* + *-kratikos* -cratic] (1602) **1** : belonging to, having the qualities of, or favoring aristocracy **2** *a* : socially exclusive (an ~ neighborhood) **b** : SNOBBISH — **aris·to·crat·i·cal·ly** \-i-k(ə-)lē\ *adv*

Ar·is·to·te·lian *or* **Ar·is·to·te·lean** \ˌar-ə-stə-ˈtē-lē-yən\ *adj* [L *Aristoteles*] (1607) : of or relating to the Greek philosopher Aristotle or his philosophy — **Aristotelian** *n* — **Ar·is·to·te·lian·ism** \-yə-ˌniz-əm\ *n*

arith·me·tic \ə-ˈrith-mə-ˌtik\ *n* [ME *arsmetrik*, fr. OF *arismetique*, fr. L *arithmetica*, fr. Gk *arithmētikē*, fr. fem. of *arithmētikos* arithmetical, fr. *arithmein* to count, fr. *arithmos* number; akin to OE *rīm* number, Gk *arariskein* to fit] (13c) **1** *a* : a branch of mathematics that deals with the nonnegative real numbers including sometimes the transfinite cardinals and with the application of the operations of addition, subtraction, multiplication, and division to them **b** : a treatise on arithmetic **2** : COMPUTATION, CALCULATION — **ar·ith·met·ic** \ˌar-ith-ˈmet-ik\ *or* **ar·ith·met·i·cal** \-i-kəl\ *adj* — **ar·ith·met·i·cal·ly** \-i-k(ə-)lē\ *adv*

arith·me·ti·cian \ə-ˌrith-mə-ˈtish-ən\ *n*

arithmetic mean *n* (1767) : a value that is computed by dividing the sum of a set of terms by the number of terms

arithmetic progression *n* (1594) : a progression (as 3, 5, 7, 9) in which the difference between any term and its predecessor is constant

-ar·i·um \ˈar-ē-əm, ˈer-\ *n suffix, pl* **-ariums** *or* **-ar·ia** \-ē-ə\ [L, fr. neut. of *-arius* -ary] : thing or place relating to or connected with (planet*arium*)

RCL011392

Case 5:03-cv-04669-JW Document 238-4 Filed 08/27/2004 Page 6 of 11

ar·chery \'ärch-(ə-)rē\ *n* (15c) **1** : the art, practice, or skill of shooting with bow and arrow **2** : an archer's weapons **3** : a body of archers

ar·che·spo·ri·um \,är-kē-'spōr-ē-əm, -'spȯr-\ *n, pl* **-spo·ria** \-ē-ə\ [NL, fr. *arche-* (as in *archegonium*) + *sporium* (fr. *spora* spore)] (1882) : the cell or group of cells from which spore mother cells develop — **ar·che·spo·ri·al** \,är-kē-'spōr-ē-əl, -'spȯr-\ *adj*

ar·che·type \'är-kē-,tīp\ *n* [*L archetypum*, fr. Gk *archetypon*, fr. neut. of *archetypos* archetypal, fr. *archein* + *typos* type] (1605) **1** : the original pattern or model of which all things of the same type are representations or copies : PROTOTYPE; *also* : a perfect example **2** : IDEA 1a, 3 **3** : an inherited idea or mode of thought in the psychology of C. G. Jung that is derived from the experience of the race and is present in the unconscious of the individual — **ar·che·typ·al** \,är-kē-'tī-pəl\ *also* **ar·che·typ·i·cal** \-'tip-i-kəl\ *adj* — **ar·che·typ·al·ly** \-pə-lē\ *adv*

arch·fiend \'ärch-'fēnd\ *n* (1667) : a chief fiend; *esp* : SATAN

archi- or **arch-** *prefix* [F or L; F, fr. L, fr. Gk — more at ARCH-] **1** : principal : chief : first (*archiblast*) **2** : primitive : original : primary (*archimycetes*) (*archicarp*)

ar·chi·carp \'är-kē-,kärp\ *n* (ca. 1887) : the female sex organ in ascomycetous fungi consisting usu. of a filamentous trichogyne and a basal fertile ascogonium

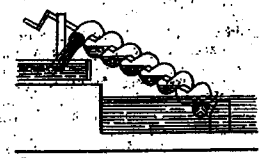
ar·chi·di·ac·o·nal \,är-kē-dī-'ak-ən-ᵊl\ *adj* [LL *archidiaconus* archdeacon] (15c) : of or relating to an archdeacon

ar·chi·epis·co·pal \,är-kē-ə-'pis-kə-pəl\ *adj* [ML *archiepiscopalis*, fr. LL *archiepiscopus* archbishop — more at ARCHBISHOP] (1611) : of or relating to an archbishop — **ar·chi·epis·co·pal·ly** \-p(ə-)lē\ *adv* — **ar·chi·epis·co·pate** \-pət, -,pāt\ *n*

ar·chil \'är-chəl\ *n* [ME *orchell*] (15c) **1** : a violet dye obtained from archil lichens (genera *Roccella* and *Lecanora*) **2** : a lichen that yields archil

ar·chi·man·drite \,är-kə-'man-,drīt\ *n* [LL *archimandrites*, fr. LGk *archimandrites*, fr. Gk *archi-* + LGk *mandra* monastery, fr. Gk, fold, pen] (1591) : a dignitary in an Eastern church ranking below a bishop; *specif* : the superior of a large monastery or group of monasteries

Ar·chi·me·des' screw \,är-kə-,mēdēz\ *n* [*Archimedes*] (ca. 1864) : a device made of a tube bent spirally around an axis or of a broad threaded screw encased by a cylinder and used to raise water



Archimedes' screw

ar·chi·pel·a·go \,är-kə-'pel-ə-,gō, ,är-chə-\ *n, pl* **-goes** *or* **-gos** [*Archipelago* Aegean sea, fr. It *Arcipelago*, lit., chief sea, fr. *arci-* (fr. L *archi-*) + Gk *pelagos* sea — more at FLAKE] (1589) **1** : an expanse of water with many scattered islands **2** : a group of islands

ar·chi·tect \'är-kə-,tekt\ *n* [MF *architecte*, fr. L *architectus*, fr. Gk *architektōn* master builder, fr. *archi-* + *tektōn* builder, carpenter — more at TECHNICAL] (1563) **1** : one who designs buildings and advises in their construction **2** : one who plans and achieves a difficult objective ⟨the great ~ of the military victory —*Time*⟩

ar·chi·tec·ton·ic \,är-kə-,tek-'tän-ik\ *adj* [L *architectonicus*, fr. Gk *architektonikos*, fr. *architektōn*] (1645) **1** : of, relating to, or according with the principles of architecture : ARCHITECTURAL **2** : having an organized and unified structure or concept that suggests an architectural design — **ar·chi·tec·ton·i·cal·ly** \-i-k(ə-)lē\ *adv*

ar·chi·tec·ton·ics \-'tän-iks\ *n pl but sing or pl in constr, also* **ar·chi·tec·ton·ic** \-ik\ (1660) **1** : the science of architecture **2** *a* : the unifying structural design of something *b* : the system of structure

ar·chi·tec·tur·al \,är-kə-'tek-chə-rəl, -'tek-shrəl\ *adj* (1762) **1** : of, relating to, or conforming to the rules of architecture **2** : having or conceived as having a single unified overall design, form, or structure — **ar·chi·tec·tur·al·ly** \-ē\ *adv*

ar·chi·tec·ture \'är-kə-,tek-chər\ *n* (1555) **1** : the art or science of building; *specif* : the art or practice of designing and building structures and esp. habitable ones **2** *a* : formation or construction as or as if as the result of conscious act *b* : a unifying or coherent form or structure ⟨the novel lacks ~⟩ **3** : architectural product or work **4** : a method or style of building

ar·chi·trave \'är-kə-,trāv\ *n* [MF, fr. Olt, fr. *archi-* + *trave* beam, fr. L *trabs* — more at THORP] (1563) **1** : the lowest division of an entablature resting in classical architecture immediately on the capital of the column — see ENTABLATURE illustration **2** : the molding around a rectangular opening (as a door)

ar·chi·val \,är-'kī-vəl\ *adj* (ca. 1828) : relating to, contained in, or constituting archives

ar·chive \'är-,kīv\ *n* [F & L; F, fr. L *archivum*, fr. Gk *archeion* government house (in pl., official documents), fr. *archē* rule, government — more at ARCH-] (1603) : a place in which public records or historical documents are preserved; *also* : the material preserved — often used in *pl*

archive *vt* **archived; archiving** (ca. 1934) : to file or collect (as records or documents) in or as if in an archive

ar·chi·vist \'är-kə-vəst, -,kī-\ *n* (1753) : a person in charge of archives

ar·chi·volt \'är-kə-,vōlt\ *n* [It *archivolto*, fr. ML *archivoltum*] (ca. 1731) : an ornamental molding around an arch corresponding to an architrave

ar·chon \'är-,kän, -kən\ *n* [L, fr. Gk *archōn*, fr. prp. of *archein*] (1579) **1** : a chief magistrate in ancient Athens **2** : a presiding officer

arch·priest \'ärch-'prēst\ *n* (14c) : a priest of prominent rank

arch·way \'ärch-,wā\ *n* (1802) : a way or passage under an arch; *also* : an arch over a passage

-arch·y \,är-kē, *in a few words also* ər-kē\ *n comb form, pl* **-ar·chies** [ME *-archie*, fr. MF, fr. L *-archia*, fr. Gk, fr. *archein* to rule — more at ARCH-] : rule : government ⟨squirearchy⟩

arc lamp *n* (1882) : an electric lamp that produces light by an arc made when a current passes between two incandescent electrodes surrounded by gas — called also *arc light*

arco \'är-(,)kō\ *adv or adj* [It, fr. *arco* bow, fr. L *arcus*] (1806) : with the bow — used as a direction in music for players of stringed instruments; compare PIZZICATO

arc·se·cant \(,)ärk-'sē-,kant, -kənt\ *n* (ca. 1961) : the inverse function to the secant ⟨if y is the secant of θ, then θ is the ~ of y⟩

arc·sine \-'sīn\ *n* (ca. 1909) : the inverse function to the sine ⟨if y is the sine of θ, then θ is the ~ of y⟩

arc·tan·gent \-'tan-jənt\ *n* (ca. 1909) : the inverse function to the tangent ⟨if y is the tangent of θ, then θ is the ~ of y⟩

¹arc·tic \'ärk-tik, 'ärt-ik\ *adj* [ME *artik*, fr. L *arcticus*, fr. Gk *arktikos*, fr. *arktos* bear, Ursa Major, north; akin to L *ursus* bear] (14c) **1** *often cap* : of or relating to the region around the north pole to approximately 65° N **2** *a* : bitter cold : FRIGID *b* : cold in temper or mood ⟨an ~ smile⟩ — **arc·ti·cal·ly** \-(t)i-k(ə-)lē\ *adv*

²arc·tic \'ärt-ik, 'ärk-tik\ *n* (1867) : a rubber overshoe reaching to the ankle or above

arctic char *n* (1902) : a char (*Salvelinus alpinus*) esp. of arctic lakes and streams of No. America

arctic circle *n, often cap A&C* (1622) : the parallel of latitude that is approximately 66½ degrees north of the equator and that circumscribes the northern frigid zone

arctic fox *n* (1772) : a small fox (*Alopex lagopus*) of the arctic regions

arctic tern *n* (1844) : a tern (*Sterna paradisaea*) that breeds in arctic regions and migrates to southern Africa and So. America

Arc·turus \ärk-'t(y)ůr-əs\ *n* [L, fr. Gk *Arktouros*, lit., bear watcher] : a giant fixed star of the first magnitude in Boötes

ar·cu·ate \'är-kyə-wət, -,wāt\ *adj* [L *arcuatus*, pp. of *arcuare* to bend like a bow, fr. *arcus* bow] (1626) : curved like a bow ⟨an ~ cloud⟩ ⟨an ~ view of a leaf⟩ — **ar·cu·ate·ly** *adv*

-ard \ərd\ *also* **-art** \ərt\ *n suffix* [ME, fr. OF, of Gmc origin; akin to OHG *-hart* (in personal names such as *Gērhart* Gerard), OE *heard* hard] : one that is characterized by performing some action, possessing some quality, or being associated with some thing esp. conspicuously or excessively ⟨braggart⟩ ⟨dullard⟩ ⟨pollard⟩

ar·deb \'är-,deb\ *n* [Ar *ardabb, irdabb*] (1836) : any of numerous Egyptian units of capacity; *esp* : the customs unit equal to 5.44 imperial or 5.619 U.S. bushels or 198.0 liters

ar·dent \'ärd-ᵊnt\ *adj* [ME, fr. MF, fr. L *ardent-, ardens*, prp. of *ardēre*] (14c) **1** : characterized by warmth of feeling typically expressed in eager zealous support or activity **2** : FIERY, HOT ⟨an ~ sun⟩ **3** : SHINING, GLOWING ⟨~ eyes⟩ *syn* see IMPASSIONED — **ar·dent·ly** \-ᵊnt-lē\ *adv*

ardent spirits *n pl* (1833) : strong distilled liquors

ar·dor \'ärd-ər\ *n* [ME *ardour*, fr. MF & L; MF, fr. L *ardor*, fr. *ardēre* to burn; akin to OHG *essa* forge, L *aridus* dry] (14c) **1** *a* : an often restless or transitory warmth of feeling ⟨the sudden ~s of youth⟩ *b* : extreme vigor or energy : INTENSITY *c* : ZEAL, LOYALTY **2** : strong or burning heat *syn* see PASSION

ar·dour *chiefly Brit var of* ARDOR

ar·du·ous \'ärj-(ə-)wəs\ *adj* [L *arduus* high, steep, difficult; akin to ON *ǫrtiigr* high, steep] (1538) **1** *a* : hard to accomplish or achieve : DIFFICULT ⟨years of ~ training⟩ *b* : marked by great labor or effort : STRENUOUS ⟨a life of ~ toil —A. C. Cole⟩ **2** : hard to climb : STEEP *syn* see HARD — **ar·du·ous·ly** *adv* — **ar·du·ous·ness** *n*

¹are [ME, fr. OE *earun*; akin to ON *eru, erum* are, OE *is* is] *pres 2d sing or pres pl of* BE

²are \'a(ə)r, 'e(ə)r, 'ār\ *n* [F, fr. L *area*] (ca. 1819) — see METRIC SYSTEM table

ar·ea \'ar-ē-ə, 'er-, 'ā-rē-ə\ *n* [L, piece of level ground, threshing floor, fr. *arēre* to be dry; akin to L *ardor*] (1538) **1** : a level piece of ground **2** : the surface included within a set of lines; *specif* : the number of unit squares equal in measure to the surface — see METRIC SYSTEM table, WEIGHT table **3** : AREAWAY ⟨went down the steps into the ~ of a house —James Joyce⟩ **4** : a particular extent of space or surface or one serving a special function **5** : the scope of a concept, operation, or activity : FIELD ⟨the whole ~ of foreign policy⟩ **6** : a part of the cerebral cortex having a particular function — **ar·e·al** \-ē-əl\ *adj* — **ar·e·al·ly** \-ə-lē\ *adv*

area code *n* (1964) : a 3-digit number that identifies each telephone service area in a country (as the U.S. or Canada)

area·way \'ar-ē-ə-,wā, 'er-, ā-rē-\ *n* (1899) : a sunken space affording access, air, and light to a basement

are·ca \ə-'rē-kə, 'ar-i-kə\ *n* [NL, fr. Pg, fr. Malayalam *atekka*] (1510) : any of several tropical Asian palms (*Areca* or related genera); *esp* : BETEL PALM

are·co·line \ə-'rek-ə-,lēn\ *n* [ISV *areca* + *-ol* + *-ine*] (1899) : a toxic parasympathomimetic alkaloid $C_8H_{13}NO_2$ that is used as a veterinary anthelmintic and occurs naturally in betel nuts

are·na \ə-'rē-nə\ *n* [L *harena, arena* sand, sandy place] (1600) **1** : an area in a Roman amphitheater for gladiatorial combats **2** *a* : an enclosed area used for public entertainment *b* : a building containing an arena **3** : a sphere of interest, activity, or competition : SCENE ⟨the political ~⟩

ar·e·na·ceous \,ar-ə-'nā-shəs\ *adj* [L *arenaceus*, fr. *arena*] (1646) **1** : resembling, made of, or containing sand or sandy particles **2** : growing in sandy places

arena theater *n* (1943) : a theater in which the stage is located in the center of the auditorium — called also *theater-in-the-round*

ar·e·nic·o·lous \,ar-ə-'nik-ə-ləs\ *adj* [L *arena* + E *-i-* + *-colous*] (ca. 1851) : living, burrowing, or growing in sand

aren't \(')ärnt, 'är-ənt\ **1** : are not **2** : am not — used in questions

ar·eo·centric \,ar-ē-ō-'sen-trik\ *adj* [Gk *Areōs* of Ares, fr. *Arēs*] (1877) : having or relating to the planet Mars as a center

are·o·la \ə-'rē-ə-lə\ *n, pl* **-lae** \-,lē\ *or* **-las** [NL, fr. L, small open space, dim. of *area*] (1664) : a small area between things or about something; *esp* : a colored ring (as about the nipple, a vesicle, or a pustule) — **are·o·lar** \-lər\ *adj* — **are·o·late** \-lət\ *adj*

ar·e·ole \'ar-ē-,ōl\ *n* (ca. 1934) : a small pit or cavity

Ar·e·op·a·gite \,ar-ē-'äp-ə-,jīt, -gīt\ *n* (14c) : a member of the Areopagus — **Ar·e·op·a·git·ic** \-,äp-ə-'jit-ik\ *adj*

RCL011393

Case 5:03-cv-04669-JW Document 238-4 Filed 08/27/2004 Page 7 of 11

independent *n* (1644) 1 *cap* : a sectarian of an English religious movement for congregational autonomy originating in the late 16th century, giving rise to Congregationalists, Baptists, and Friends, and forming one of the major political groupings of the period of Cromwell 2 : one that is independent; *esp, often cap* : one that is not bound by or definitively committed to a political party

independent assortment *n* (ca. 1948) : formation of random combinations of chromosomes in meiosis and of genes on different pairs of homologous chromosomes by the passage at random of one of each diploid pair of homologous chromosomes into each gamete independently of each other pair

independent variable *n* (1852) : a mathematical variable whose value is specified first and determines the value of one or more other values in an expression or function ⟨in $z = x^2 + 3xy + y^2$, x and y are *independent variables*⟩

in-depth \(ˌ)in-ˈdepth\ *adj* (1965) : COMPREHENSIVE, THOROUGH ⟨an ~ study⟩

in-de-scrib-able \ˌin-di-ˈskrī-bə-bəl\ *adj* (1794) 1 : that cannot be described ⟨an ~ sensation⟩ 2 : surpassing description ⟨~ joy⟩ — **in-de-scrib-able-ness** *n* — **in-de-scrib-ably** \-blē\ *adv*

in-de-struc-ti-ble \-ˈstrək-tə-bəl\ *adj* [prob. fr. LL *indestructibilis*, fr. L *in-* + *destructus*, pp. of *destruere* to tear down — more at DESTROY] (1667) : not destructible — **in-de-struc-ti-bil-i-ty** \-ˌstrək-tə-ˈbil-ət-ē\ *n* — **in-de-struc-ti-ble-ness** \-ˈstrək-tə-bəl-nəs\ *n* — **in-de-struc-ti-bly** \-blē\ *adv*

in-de-ter-min-able \ˌin-di-ˈtərm-(ə-)nə-bəl\ *adj* (15c) 1 : incapable of being definitely decided or settled 2 : incapable of being definitely fixed or ascertained — **in-de-ter-min-ably** \-blē\ *adv*

in-de-ter-mi-na-cy \-ˈtərm-(ə-)nə-sē\ *n* (1649) : the quality or state of being indeterminate

indeterminacy principle *n* (ca. 1928) : UNCERTAINTY PRINCIPLE

in-de-ter-mi-nate \ˌin-di-ˈtərm-(ə-)nət\ *adj* [ME *indeterminat*, fr. LL *indeterminatus*, fr. L *in-* + *determinatus*, pp. of *determinare* to determine] (14c) 1 a : not definitely or precisely determined or fixed : VAGUE b : not known in advance c : not leading to a definite end or result 2 : having an infinite number of solutions ⟨a system of ~ equations⟩ 3 : being one of the seven undefined mathematical expressions

$$\frac{0}{0},\ \frac{\infty}{\infty},\ \infty \cdot 0,\ 1^{\infty},\ 0^{0},\ \infty^{0},\ \infty - \infty$$

4 : RACEMOSE — **in-de-ter-mi-nate-ly** *adv* — **in-de-ter-mi-nate-ness** *n* — **in-de-ter-mi-na-tion** \-ˌtər-mə-ˈnā-shən\ *n*

in-de-ter-min-ism \-ˈtər-mə-ˌniz-əm\ *n* (1874) 1 a : a theory that the will is free and that deliberate choice and actions are not determined by or predictable from antecedent causes b : a theory that holds that not every event has a cause 2 : the quality or state of being indeterminate; *esp* : UNPREDICTABILITY — **in-de-ter-min-ist** \-ˈtərm-(ə-)nəst\ *n* — **in-de-ter-min-is-tic** \-ˌtər-mə-ˈnis-tik\ *adj*

in-dex \ˈin-ˌdeks\ *n, pl* **in-dex-es** *or* **in-di-ces** \-də-ˌsēz\ [L *indic-, index*, fr. *indicare* to indicate] (1571) 1 a : a device (as the pointer on a scale or the gnomon of a sundial) that serves to indicate a value or quantity b : something (as a physical feature or a mode of expression) that leads one to a particular fact or conclusion : INDICATION ⟨the fertility of the land is an ~ of the country's wealth⟩ 2 : a list (as of bibliographical information or citations to a body of literature) arranged usu. in alphabetical order of some specified datum (as author, subject, or keyword): as a : a list of items (as topics or names) treated in a printed work that gives for each item the page number where it may be found b : THUMB INDEX c : a bibliographical analysis of groups of publications that is usu. published periodically 3 : a list of restricted or prohibited material; *specif, cap* : a list of books the reading of which is prohibited or restricted for Roman Catholics by the church authorities 4 *pl usu indices* : a number or symbol or expression (as an exponent) associated with another to indicate a mathematical operation to be performed or to indicate use or position in an arrangement ⟨the *indices* 2 and 3 locate the element a_{23} in the second row and third column of a determinant⟩ 5 : a character ☞ used to direct attention to a note or paragraph — called also *fist* 6 a : a number (as a ratio) derived from a series of observations and used as an indicator of measure (as of a condition, property, or phenomenon); *specif* : INDEX NUMBER b : the ratio of one dimension of a thing (as an anatomical structure) to another dimension — **in-dex-i-cal** \ˌin-ˈdek-si-kəl\ *adj*

index *vt* (1720) 1 a : to provide with an index b : to list in an index 2 : to serve as an index of 3 : to regulate (as wages, prices, or interest rates) by indexation ~ *vi* : to index something — **in-dex-er** *n*

index-ation \ˌin-dek-ˈsā-shən\ *n* (1960) : a system of economic control in which certain variables (as wages and interest) are tied to a cost-of-living index so that both rise or fall at the same rate and the detrimental effect of inflation is theoretically eliminated

index finger *n* (1849) : FOREFINGER

index fossil *n* (1900) : a fossil usu. with a narrow time range and wide spatial distribution that is used in the identification of related geologic formations

indexing *n* (ca. 1974) : INDEXATION

index number *n* (ca. 1896) : a number used to indicate change in magnitude (as of cost or price) as compared with the magnitude at some specified time usu. taken as 100

index of refraction (ca. 1829) : the ratio of the velocity of radiation (as light) in the first of two media to its velocity in the second as it passes from one into the other

India — see IND-

India \ˈin-dē-ə\ (ca. 1952) — a communications code word for the letter *i*

India ink *n, often cap I* (1665) 1 : a solid black pigment (as specially prepared lampblack) used in drawing and lettering 2 : a fluid ink containing usu. of a fine suspension of india ink in a liquid

India-man \ˈin-dē-ə-mən\ *n* (1709) : a merchant ship formerly used in trade with India; *esp* : a large sailing ship used in this trade

Indian \ˈin-dē-ən *also* ˈin-din *or chiefly dial* -jən\ *n* (14c) 1 : a native or inhabitant of the subcontinent of India or of the East Indies 2 : a the belief held by Columbus that the lands he discovered were part of Asia] : AMERICAN INDIAN b : one of the native languages of American Indians — **Indian** *adj* — **In-di-an-ness** *n*

Indian agent *n* (1883) : an official representative of the U.S. federal government to American Indian tribes esp. on reservations

Indian club *n* (1857) : a usu. wooden club shaped like a large bottle or tenpin that is swung for gymnastic exercise

Indian corn *n* (1617) 1 : a tall widely cultivated American cereal grass (*Zea mays*) bearing seeds on elongated ears 2 : the ears of Indian corn; *also* : its edible seeds

Indian elephant *n* (1607) : ELEPHANT 1b

Indian file *n* (1758) : SINGLE FILE

Indian giver *n* (ca. 1848) : one that gives something to another and then takes it back or expects an equivalent in return — sometimes taken to be offensive — **Indian giving** *n*

Indian hemp *n* (1619) 1 : an American dogbane (*Apocynum cannabinum*) with milky juice, tough fibrous bark, and an emetic and cathartic root 2 : HEMP 1

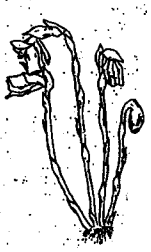
In-di-an-ism \ˈin-dē-ə-ˌniz-əm\ *n* (1651) 1 : the qualities or culture distinctive of Indians 2 : policy designed to further the interests or culture of Indians — **In-di-an-ist** \-nəst\ *adj or n*

Indian licorice *n* (ca. 1890) : ROSARY PEA 1

Indian meal *n* (1609) : CORNMEAL

Indian paintbrush *n* (1892) 1 : any of a genus (*Castilleja*) of herbaceous plants of the figwort family that have brightly colored bracts — called also *painted cup* 2 : ORANGE HAWKWEED

Indian pipe *n* (1817) : a waxy white leafless saprophytic herb (*Monotropa uniflora* of the family Monotropaceae, the Indian-pipe family) of Asia and the U.S.



Indian pipe

Indian pudding *n* (1722) : a baked pudding made chiefly of cornmeal, milk, and molasses

Indian red *n* (ca. 1753) 1 a : a yellowish red earth containing hematite and used as a pigment b : any of various light red to purplish brown pigments made by calcining iron salts 2 : a strong or moderate reddish brown

Indian sign *n* (1910) : HEX, SPELL

Indian summer *n* (1778) 1 : a period of warm or mild weather in late autumn or early winter 2 : a happy or flourishing period occurring toward the end of something ⟨life in the *Indian summer* of Czarist Russia —John Davenport⟩

Indian tobacco *n* (1618) 1 : an American wild lobelia (*Lobelia inflata*) with small blue flowers 2 : a wild tobacco (*Nicotiana bigelovii*) found in dry valleys from southern California to southern Oregon

In-di-an-wres-tle \ˈin-dē-ən-ˌres-əl, -ˌras-\ *vi* [back-formation fr. *Indian wrestling*] (1938) : to engage in Indian wrestling

Indian wrestling *n* (1913) 1 : wrestling in which two people lie side by side on their backs in reversed position locking their near arms and raising and locking the corresponding legs and attempt to force each other's leg down and turn the other wrestler on his face 2 : wrestling in which two people stand face to face gripping usu. their right hands and setting the outsides of the corresponding feet together and attempt to force each other off balance 3 : ARM WRESTLING

India paper *n* (1768) 1 : a thin absorbent paper used esp. for proving inked intaglio surfaces (as steel engravings) 2 : a thin tough opaque printing paper

India rubber *n, often cap I* (1790) : RUBBER 2a

In-dic \ˈin-dik\ *adj* (1877) 1 : of or relating to the subcontinent of India : INDIAN 2 : of, relating to, or constituting the Indian branch of the Indo-European languages — see INDO-EUROPEAN LANGUAGES table — **Indic** *n*

in-di-can \ˈin-də-ˌkan\ *n* [L *indicum* indigo — more at INDIGO] (1859) 1 : an indigo-forming substance $C_8H_7NO_4S$ found as a salt in urine and other animal fluids; *also* : its potassium salt $C_8H_6KNO_4S$ 2 : a glucoside $C_{14}H_{17}NO_6$ occurring esp. in the indigo plant and being a source of natural indigo

in-di-cant \ˈin-di-kənt\ *n* (1623) : something that serves to indicate

in-di-cate \ˈin-də-ˌkāt\ *vt* **-cat-ed; -cat-ing** [L *indicatus*, pp. of *indicare*, fr. *in-* + *dicare* to proclaim, dedicate — more DICTION] (1651) 1 a : to point out or point to b : to be a sign, symptom, or index of ⟨the high fever ~s a serious condition⟩ c : to demonstrate or suggest the necessity or advisability of ⟨*Indicated* the need for a new school⟩ 2 : to state or express briefly : SUGGEST ⟨*Indicated* his desire to cooperate⟩

in-di-ca-tion \ˌin-də-ˈkā-shən\ *n* (15c) 1 a : something that serves to indicate b : something that is indicated as advisable or necessary 2 : the action of indicating 3 : the degree indicated on a graduated instrument : READING — **in-di-ca-tion-al** \-shnəl, -shən-ᵊl\ *adj*

in-dic-a-tive \ən-ˈdik-ət-iv\ *adj* (15c) 1 : of, relating to, or constituting a verb form or set of verb forms that represents the denoted act or state as an objective fact ⟨the ~ mood⟩ 2 : serving to indicate ⟨actions ~ of fear⟩ — **in-dic-a-tive-ly** *adv*

²indicative *n* (1530) 1 : the indicative mood of a language 2 : a form in the indicative mood

in-di-ca-tor \ˈin-də-ˌkāt-ər\ *n* (1660) 1 : one that indicates: as a : an index hand (as on a dial) : POINTER b (1) : a pressure gauge (2) : an instrument for automatically making a diagram that indicates the pressure in and volume of the working fluid of an engine throughout the cycle c : a dial that registers something (as the movement of an elevator) 2 a : a substance (as litmus) used to show visually (as by change of color) the condition of a solution with respect to the presence of a particular material (as a free acid or alkali) b : TRACER 4b 3 : an organism or ecological community so strictly associated with particular environmental conditions that its presence is indicative of the existence of these conditions 4 : any of a group of statistical values (as

RCL011395

Case 5:03-cv-04669-JW Document 238-4 Filed 08/27/2004 Page 8 of 11

in-cur-rence \in-'kər-ən(t)s, -'kə-rən(t)s\ *n* (1656) : the act or process of incurring

in-cur-rent \-ənt, -rənt\ *adj* [L *incurrent-, incurrens*, prp. of *incurrere*] (1851) : giving passage to a current that flows inward

in-cur-sion \in-'kər-zhən\ *n* [ME, fr. MF or L; MF, fr. L *incursion-, incursio*, fr. *incursus*, pp. of *incurrere*] (15c) 1 : a hostile entrance into a territory : RAID 2 : an entering in or into (as an activity or undertaking) ⟨his only ~ into the arts⟩

in-cur-vate \in-,kər-,vāt, ('')in-'kər-\ *vt* **-vat-ed; -vat-ing** (1578) : to cause to curve inward : BEND — **in-cur-vate** \in-,kər-,vāt, ('')in-'kər-vət\ *adj* — **in-cur-va-tion** \,in-,kər-'vā-shən\ *n* — **in-cur-va-ture** \('')in-'kər-və-,chu(ə)r, -,chər, -,t(y)u(ə)r\ *n*

in-curve \('')in-'kərv, 'in-,\ *vt* [L *incurvare*, fr. *in-* + *curvare* to curve, fr. *curvus* curved — more at CROWN] (15c) : to bend so as to curve inward

in-cus \'iŋ-kəs\ *n, pl* **in-cu-des** \iŋ-'kyü-(,)dēz, 'iŋ-kyə-,dēz\ [NL, fr. L, anvil, fr. *incudere*] (1669) : the middle of a chain of three small bones in the ear of a mammal — called also *anvil*; see EAR illustration

in-cuse \in-'kyüz, -'kyüs\ *adj* [L *incusus*, pp. of *incudere* to stamp, strike, fr. *in-* + *cudere* to beat — more at HEW] (1818) : formed by stamping or punching in — used chiefly of old coins or features of their design

Ind \'ind, 'ind\ *n* (13c) 1 *archaic* : INDIA 2 *obs* : INDIES

ind- *or* **indi-** *or* **indo-** *comb form* [ISV, fr. L *indicum* — more at INDIGO] 1 : indigo ⟨*indoxyl*⟩. 2 : resembling indigo (as in color) ⟨*indophenol*⟩

in-da-ba \in-'däb-ə\ *n* [Zulu *in-daba* affair] *chiefly So Afr* (1827) : CONFERENCE, PARLEY

in-da-gate \'in-də-,gāt\ *vt* **-gat-ed; -gat-ing** [L *indagatus*, pp. of *indagare*, fr. *indago* act of enclosing, investigation, fr. OL *indu* in + L *agere* to drive — more at INDIGENOUS, AGENT] (1623) : to search into : INVESTIGATE — **in-da-ga-tion** \,in-də-'gā-shən\ *n* — **in-da-ga-tor** \'in-də-,gāt-ər\ *n*

ind-amine \'in-də-,mēn\ *n* [ISV] (1888) : any of a series of organic bases of which the simplest has the formula $C_{12}H_{11}N_3$ and which form salts that are unstable blue and green dyes

in-debt-ed \in-'det-əd\ *adj* [ME *indetted*, fr. OF *endeté*, pp. of *endeter* to involve in debt, fr. *en-* + *dete* debt] (13c) 1 : owing money 2 : owing gratitude or recognition to another : BEHOLDEN

in-debt-ed-ness *n* (1647) 1 : the condition of being indebted 2 : something (as an amount of money) that is owed

in-de-cen-cy \('')in-'dē-sᵊn-sē\ *n* (1589) 1 : the quality or state of being indecent 2 : something (as a word or action) that is indecent

in-de-cent \-sᵊnt\ *adj* [MF or L; MF *indécent*, fr. L *indecent-, indecens*, fr. *in-* + *decent-, decens* decent] (1563) : not decent; *esp* : grossly unseemly or offensive to manners or morals *syn* see INDECOROUS — **in-de-cent-ly** *adv*

indecent assault *n* (1861) : an immoral act or series of acts exclusive of rape committed against another person without consent

indecent exposure *n* (1851) : intentional exposure of part of one's body (as the genitals) in a place where such exposure is likely to be an offense against the generally accepted standards of decency

in-de-ci-pher-able \,in-di-'sī-f(ə-)rə-bəl\ *adj* (1802) : incapable of being deciphered

in-de-ci-sion \,in-di-'sizh-ən\ *n* [F *indécision*, fr. *indécis* undecided, fr. LL *indecisus*, fr. L *in-* + *decisus*, pp. of *decidere* to decide] (ca. 1763) : a wavering between two or more possible courses of action : IRRESOLUTION

in-de-ci-sive \,in-di-'sī-siv\ *adj* (1726) 1 : not decisive : INCONCLUSIVE 2 : marked by or prone to indecision : IRRESOLUTE 3 : not clearly marked out : INDEFINITE — **in-de-ci-sive-ly** *adv* — **in-de-ci-sive-ness** *n*

in-de-clin-able \,in-di-'klī-nə-bəl\ *adj* [MF, fr. LL *indeclinabilis*, fr. L *in-* + LL *declinabilis* capable of being inflected, fr. L *declinare* to inflect — more at DECLINE] (14c) : having no grammatical inflections

in-de-com-pos-able \,in-,dē-kəm-'pō-zə-bəl\ *adj* (1807) : not capable of being separated into component parts or elements

in-de-co-rous \('')in-'dek-(ə-)rəs; ,in-di-'kōr-əs, -'kȯr-\ *adj* [L *indecorus*, fr. *in-* + *decorus* decorous] (1682) : not decorous — **in-de-co-rous-ly** *adv* — **in-de-co-rous-ness** *n*

syn INDECOROUS, IMPROPER, UNSEEMLY, INDECENT, UNBECOMING, INDELICATE mean not conforming to what is accepted as right, fitting, or in good taste. INDECOROUS suggests a violation of accepted standards of good manners; IMPROPER applies to a broader range of transgressions of rules not only of social behavior but of ethical practice or logical procedure or prescribed method; UNSEEMLY adds a suggestion of special inappropriateness to a situation or an offensiveness to good taste; INDECENT implies great unseemliness or gross offensiveness esp. in referring to sexual matters; UNBECOMING suggests behavior or language that does not suit one's character or status; INDELICATE implies a lack of modesty or of tact or of refined perception of feeling

in-de-co-rum \,in-di-'kōr-əm, -'kȯr-\ *n* [L, neut. of *indecorus*] (1575) 1 : something that is indecorous 2 : lack of decorum : IMPROPRIETY

in-deed \in-'dēd\ *adv* (14c) 1 : without any question : TRULY; UNDENIABLY — often used interjectionally to express irony or disbelief or surprise 2 : in reality 3 : all things considered : as a matter of fact

in-de-fat-i-ga-ble \,in-di-'fat-i-gə-bəl\ *adj* [MF, fr. L *indefatigabilis*, fr. *in-* + *defatigare* to fatigue, fr. *de-* down + *fatigare* to fatigue — more at DE-] (1586) : incapable of being fatigued : UNTIRING — **in-de-fa-ti-ga-bil-i-ty** \-,fat-i-gə-'bil-ət-ē\ *n* — **in-de-fat-i-ga-ble-ness** \-'fat-i-gə-bəl-nəs\ *n* — **in-de-fat-i-ga-bly** \-blē\ *adv*

in-de-fea-si-ble \-'fē-zə-bəl\ *adj* (1548) : not capable of being annulled or voided or undone ⟨an ~ right⟩ — **in-de-fea-si-bil-i-ty** \-,fē-zə-'bil-ət-ē\ *n* — **in-de-fea-si-bly** \-'fē-zə-blē\ *adv*

in-de-fec-ti-ble \-'fek-tə-bəl\ *adj* (1659) 1 : not subject to failure or decay : LASTING 2 : free of faults : FLAWLESS — **in-de-fec-ti-bil-i-ty** \-,fek-tə-'bil-ət-ē\ *n* — **in-de-fec-ti-bly** \-'fek-tə-blē\ *adv*

in-de-fen-si-ble \-'fen(t)-sə-bəl\ *adj* (1529) 1 a : incapable of being maintained as right or valid : UNTENABLE b : incapable of being justified or excused : INEXCUSABLE 2 : incapable of being protected against physical attack — **in-de-fen-si-bil-i-ty** \-,fen(t)-sə-'bil-ət-ē\ *n* — **in-de-fen-si-bly** \-'fen(t)-sə-blē\ *adv*

in-de-fin-able \-'fī-nə-bəl\ *adj* (1690) : incapable of being precisely described or analyzed — **in-de-fin-abil-i-ty** \-,fī-nə-'bil-ət-ē\ *n* — **indefinable** *n* — **in-de-fin-able-ness** \-'fī-nə-bəl-nəs\ *n* — **in-de-fin-ably** \-blē\ *adv*

in-def-i-nite \('')in-'def-(ə-)nət\ *adj* [L *indefinitus*, fr. *in-* + *definitus* definite] (15c) : not definite: as a : typically designating an unidentified or not immediately identifiable person or thing ⟨the ~ articles *a* and *an*⟩ b : not precise : VAGUE c : having no exact limits d *of floral organs* : numerous and difficult to ascertain in number — **indefinite** *n* — **in-def-i-nite-ly** *adv* — **in-def-i-nite-ness** *n*

indefinite integral *n* (ca. 1877) : any function whose derivative is a given function

in-de-his-cent \,in-di-'his-ᵊnt\ *adj* (1832) : remaining closed at maturity ⟨~ fruits⟩ — **in-de-his-cence** \-ᵊn(t)s\ *n*

in-del-i-ble \in-'del-ə-bəl\ *adj* [ML *indelibilis*, alter. of L *indelebilis*, fr. *in-* + *delēre* to delete] (1529) 1 : that cannot be removed, washed away, or erased 2 : making marks that cannot easily be removed ⟨an ~ pencil⟩ 3 : LASTING, UNFORGETTABLE — **in-del-i-bil-i-ty** \(,)in-,del-ə-'bil-ət-ē\ *n* — **in-del-i-bly** \in-'del-ə-blē\ *adv*

in-del-i-ca-cy \-kə-sē\ *n* (1712) 1 : the quality or state of being indelicate 2 : something that is indelicate

in-del-i-cate \('')in-'del-i-kət\ *adj* (1742) : not delicate: a (1) : lacking in or offending against propriety : IMPROPER (2) : verging on the indecent : COARSE b : marked by a lack of feeling for the sensibilities of others : TACTLESS *syn* see INDECOROUS — **in-del-i-cate-ly** *adv* — **in-del-i-cate-ness** *n*

in-dem-ni-fi-ca-tion \in-,dem-nə-fə-'kā-shən\ *n* (1732) 1 a : the action of indemnifying b : the condition of being indemnified 2 : INDEMNITY 2b

in-dem-ni-fy \in-'dem-nə-,fī\ *vt* **-fied; -fy-ing** [L *indemnis* unharmed, fr. *in-* + *damnum* damage] (1611) 1 : to secure against hurt, loss, or damage 2 : to make compensation to for incurred hurt, loss, or damage *syn* see PAY — **in-dem-ni-fi-er** \-,fī(-ə)r\ *n*

in-dem-ni-ty \in-'dem-nət-ē\ *n, pl* **-ties** (15c) 1 a : security against hurt, loss, or damage b : exemption from incurred penalties or liabilities 2 a : INDEMNIFICATION 1 b : something that indemnifies

in-de-mon-stra-ble \,in-di-'män(t)-strə-bəl, (,)in-'dem-ən-strə-\ *adj* (1570) : incapable of being demonstrated : not subject to proof — **in-de-mon-stra-bly** \-blē\ *adv*

in-dene \'in-,dēn\ *n* [ISV, fr. *indole*] (1888) : a liquid hydrocarbon C_9H_8 obtained from coal tar and used esp. in making resins

¹in-dent \in-'dent\ *vb* [ME *indenten*, fr. MF *endenter*, fr. OF, fr. *en-* + *dent* tooth, fr. L *dent-, dens* — more at TOOTH] *vt* (14c) 1 a : to divide (a document) so as to produce sections with irregular edges that can be matched for authentication b : to draw up (as a deed) in two or more exactly corresponding copies 2 : INDENTURE 3 a : to notch the edge of : make jagged b : to cut into for the purpose of mortising or dovetailing 4 : to set (as a line of a paragraph) in from the margin 5 : to join together by or as if by mortises or dovetails 6 *chiefly Brit* : to order by an indent ~ *vi* 1 *obs* : to make a formal or express agreement 2 : to form an indentation 3 *chiefly Brit* : to make out an indent for something — **in-dent-er** *n* — **indent on** 1 *chiefly Brit* : to make a requisition on 2 *chiefly Brit* : to draw on

²in-dent \in-'dent, 'in-,\ *n* (15c) 1 a : INDENTURE 1 b : a certificate issued by the U.S. at the close of the American Revolution for the principal or interest on the public debt 2 *chiefly Brit* a : an official requisition b : a purchase order for goods esp. when sent from a foreign country 3 : INDENTION

³in-dent \in-'dent\ *vt* [ME *endenten*, fr. *en-* + *denten* to dent] (15c) 1 : to force inward so as to form a depression 2 : to form a dent in — **in-dent-er** *n*

⁴in-dent \in-'dent, 'in-,\ *n* (1596) : INDENTATION

in-den-ta-tion \,in-,den-'tā-shən\ *n* (1728) 1 a : an angular cut in an edge : NOTCH b : a recess in a surface 2 : the action of indenting : the condition of being indented 3 : DENT 4 : INDENTION 2b

in-den-tion \in-'den-chən\ *n* (1763) 1 *archaic* : INDENTATION 1 2 a : the action of indenting : the condition of being indented b : the blank space produced by indenting

¹in-den-ture \in-'den-chər\ *n* [ME *endenture*, fr. MF, fr. *enderter*] (14c) 1 a (1) : a document or a section of a document that is indented (2) : a formal or official document usu. executed in two or more copies (3) : a contract binding one person to work for another for a given period of time — usu. used in pl. b : a formal certificate (as an inventory or voucher) prepared for purposes of control c : a document stating the terms under which a security (as a bond) is issued 2 : INDENTATION 1 3 [*³indent*] : DENT

²indenture *vt* **in-den-tured; in-den-tur-ing** \-'dench-(ə-)riŋ\ (1676) : to bind (as an apprentice) by or as if by indentures

indentured servant *n* (1723) : a person who binds himself by indentures to work for another for a specified time esp. in return for payment of his travel expenses and maintenance

in-de-pen-dence \,in-də-'pen-dən(t)s\ *n* (1640) 1 : the quality or state of being independent 2 *archaic* : COMPETENCE 1

Independence Day *n* (1791) : a civil holiday for the celebration of the anniversary of the beginnings of national independence; *specif* : July 4 observed as a legal holiday in the U.S. in commemoration of the adoption of the Declaration of Independence in 1776

in-de-pen-den-cy \,in-də-'pen-dən-sē\ *n* (1611) 1 : INDEPENDENCE 1 2 *cap* : the Independent polity or movement 3 : an independent political unit

¹in-de-pen-dent \,in-də-'pen-dənt\ *adj* (1611) 1 : not dependent: as a (1) : not subject to control by others : SELF-GOVERNING (2) : not affiliated with a larger controlling unit b (1) : not requiring or relying on something else : not contingent ⟨an ~ conclusion⟩ (2) : not looking to others for one's opinions or for guidance in conduct (3) : not bound by or committed to a political party c (1) : not requiring or relying on others (as for care or livelihood) ⟨~ of his parents⟩ (2) : being enough to free one from the necessity of working for a living ⟨a man of ~ means⟩ d : showing a desire for freedom ⟨an ~ manner⟩ e (1) : not determined by or capable of being deduced or derived from or expressed in terms of members (as axioms or equations) of the set under consideration; *esp* : having linear independence ⟨an ~ set of vectors⟩ (2) : having the property that the joint probability (as of events or samples) or the joint probability density function (as of random variables) equals the product of the probabilities or probability density functions of separate occurrence 2 *cap* : of or relating to the Independents 3 a : MAIN 5 ⟨the ~ clause⟩ b : neither deducible from nor incompatible with another statement ⟨~ postulates⟩ *syn* see FREE

in-de-pen-dent-ly *adv*

RCL011394

Case 5:03-cv-04669-JW Document 238-4 Filed 08/27/2004 Page 9 of 11

associated in our minds with "developed" rather than "developing" states —*Times Lit. Supp.*) **2** : to be made up of ⟨a committee *comprising* three faculty members and three public members⟩ **3** : COMPOSE, CONSTITUTE ⟨a misconception as to what ~s a literary generation —William Styron⟩ ⟨about 8 percent of our military forces are *comprised* of women —Jimmy Carter⟩

usage Although it has been in use since the late 18th century, sense 3 is still attacked as wrong. Why it has been singled out is not clear, but until comparatively recent times it was found chiefly in scientific or technical writing rather than belles lettres. Our current evidence shows a slight shift in usage: sense 3 is somewhat more frequent in recent literary use than the earlier senses. You should be aware, however, that if you use sense 3 you may be subject to criticism for doing so, and you may want to choose a safer synonym such as *compose* or *make up*.

¹com-pro-mise \'käm-prə-ˌmīz\ *n* [ME, mutual promise to abide by an arbiter's decision, fr. MF *compromis*, fr. L *compromissum*, fr. neut. of *compromissus*, pp. of *compromittere* to promise mutually, fr. *com-* + *promittere* to promise — more at PROMISE] (15c) **1 a** : settlement of differences by arbitration or by consent reached by mutual concessions **b** : something blending qualities of two different things **2** : a concession to something derogatory or prejudicial ⟨a ~ of principles⟩

²compromise *vb* **-mised; -mis·ing** *vt* (1598) **1** *obs* : to bind by mutual agreement **2** : to adjust or settle by mutual concessions **3** : to expose to discredit or mischief ~ *vi* **1** : to come to agreement by mutual concession **2** : to make a shameful or disreputable concession — **com·pro·mis·er** *n*

compt \'kau̇nt; 'käm(p)t\ *archaic var of* COUNT

comp·troller \kən-'trō-lər, 'käm(p)-, ˌkäm(p)-'\ *n* [ME, alter. of *controller*] (15c) **1** : a royal household official who examines and supervises expenditures **2** : a public official who audits government accounts and sometimes certifies expenditures **3** : CONTROLLER Ic — **comp·troll·er·ship** \-ˌship\ *n*

com·pul·sion \kəm-'pəl-shən\ *n* [ME, fr. MF or LL; MF, fr. LL *compulsion-, compulsio*, fr. L *compulsus*, pp. of *compellere* to compel] (15c) **1 a** : an act of compelling : the state of being compelled **b** : a force that compels **2** : an irresistible impulse to perform an irrational act

com·pul·sive \-'pəl-siv\ *adj* (1588) **1** : having power to compel **2** : of, relating to, caused by, or suggestive of psychological compulsion or obsession ⟨~ actions⟩ — **com·pul·sive·ly** *adv* — **com·pul·sive·ness** *n* — **com·pul·siv·i·ty** \kəm-ˌpəl-'siv-ət-ē, ˌkäm-\ *n*

com·pul·sory \kəm-'pəls-(ə-)rē\ *adj* (1581) **1** : MANDATORY, ENFORCED **2** : COERCIVE, COMPELLING — **com·pul·so·ri·ly** \-(ə-)rə-lē\ *adv*

com·punc·tion \kəm-'pəŋ(k)-shən\ *n* [ME *compunccioun*, fr. MF *compunction*, fr. LL *compunction-, compunctio*, fr. L *compunctus*, pp. of *compungere* to prick hard, sting, fr. *com-* + *pungere* to prick — more at PUNGENT] (14c) **1 a** : anxiety arising from awareness of guilt ⟨~s of conscience⟩ **b** : distress of mind over an anticipated action or result ⟨he showed no ~ in planning devilish engines of . . . destruction —Havelock Ellis⟩ **2** : a twinge of misgiving ; SCRUPLE ⟨cheated without ~⟩ *syn* see PENITENCE, QUALM — **com·punc·tious** \-shəs\ *adj*

com·pur·ga·tion \ˌkäm(p)ər-'gā-shən\ *n* [LL *compurgation-, compurgatio*, fr. L *compurgatus*, pp. of *compurgare* to clear completely, fr. *com-* + *purgare* to purge] (1658) : the clearing of an accused person by oaths of persons who swear to his veracity or innocence

com·pur·ga·tor \'käm-(ˌ)pər-ˌgāt-ər\ *n* (1533) : one that under oath vouches for the character or conduct of an accused person

com·put·able \kəm-'pyüt-ə-bəl\ *adj* (1646) : capable of being computed — **com·put·abil·i·ty** \-ˌpyüt-ə-'bil-ət-ē\ *n*

com·pu·ta·tion \ˌkäm-pyu̇-'tā-shən\ *n* (15c) **1 a** : the act or action of computing ; CALCULATION **b** : the use or operation of a computer **2** : a system of reckoning **3** : an amount computed — **com·pu·ta·tion·al** \-shnəl, -shən-ᵊl\ *adj*

¹com·pute \kəm-'pyüt\ *n* (1588) : COMPUTATION

²compute *vb* **com·put·ed; com·put·ing** [L *computare* — more at COUNT] *vt* (1631) : to determine esp. by mathematical means ⟨~ your income tax⟩; *also* : to determine or calculate by means of a computer ~ *vi* **1** : to make calculation : RECKON **2** : to use a computer

computed tomography *n* (ca. 1977) : radiography in which a three-dimensional image of a body structure is constructed by computer from a series of plane cross-sectional images made along an axis — called also *computed axial tomography, computerized axial tomography, computerized tomography*

com·put·er \kəm-'pyüt-ər\ *n, often attrib* (1646) : one that computes; *specif* : a programmable electronic device that can store, retrieve, and process data — **com·put·er·like** \-ˌlīk\ *adj*

com·put·er·ese \-ˌpyüt-ə-'rēz, -'rēs\ *n* (ca. 1960) : jargon used by computer technologists

com·put·er·ise *chiefly Brit var of* COMPUTERIZE

com·put·er·ist \kəm-'pyüt-ə-rəst\ *n* (1973) : a person who uses or operates a computer

com·put·er·ize \kəm-'pyüt-ə-ˌrīz\ *vt* **-ized; -iz·ing** (1957) **1** : to carry out, control, or produce by means of a computer **2** : to equip with computers **3** : to store in a computer **b** : to put in a form that a computer can use — **com·put·er·iz·able** \-ˌrī-zə-bəl\ *adj* — **com·put·er·i·za·tion** \-ˌpyüt-ə-rə-'zā-shən\ *n*

com·put·er·nik \kəm-'pyüt-ər-ˌnik\ *n* [*computer* + *-nik*] (1968) : COMPUTERIST

com·rade \'käm-ˌrad, -rəd, *esp Brit* -ˌrād\ *n* [MF *camarade* group sleeping in one room, roommate, companion, fr. OSp *camarada*, fr. *cámara* room, fr. LL *camera, camara* — more at CHAMBER] (1544) **1 a** : an intimate friend or associate : COMPANION **b** : a fellow soldier **2** [fr. its use as a form of address by communists] : COMMUNIST — **com·rade·li·ness** \-lē-nəs\ *n* — **com·rade·ly** *adj* — **com·rade·ship** \-ˌship\ *n*

com·rade·ry \'käm-ˌrad-(ə-)rē, -rəd-rē, -ˌrād-(ə-)rē\ *n* (1879) : CAMARADERIE

Comsat \'käm-ˌsat\ *service mark* — used for communications services involving an artificial satellite

Com·stock·ery \'käm-ˌstäk-ə-rē *also* 'kəm-\ *n* [Anthony *Comstock* + E *-ery*] (1909) **1** : strict censorship of materials considered obscene **2** : censorious opposition to alleged immorality (as in literature)

Comstockian \käm-'stäk-ē-ən *also* kəm-\ *adj* (1921) : of or relating to Comstockery

com·symp \'käm-ˌsimp\ *n* [*communist* + *sympathizer*] (ca. 1961) : a person who sympathizes with communist causes — usu. used disparagingly

Comt·ism *or* **Comt·ean** \'käm(p)-tē-ən, 'kōⁿ(n)t-ē-\ *adj* (1846) : of or relating to Auguste Comte or his doctrines — **Comt·ism** \'käm(p)-ˌtiz-əm, 'kōⁿ(n)t-ˌiz-\ *n* — **Comt·ist** \'käm(p)-təst, 'kōⁿ(n)t-əst\ *adj or n*

¹con \'kän\ *vt* **conned; con·ning** [ME *connen* to know; learn, study, alter. of *cunnen* to know, infin. of *can* — more at CAN] (13c) **1** : to commit to memory **2** : to study or examine closely : PERUSE

²con *var of* CONN

³con *adv* [ME, short for *contra*] (15c) : on the negative side : in opposition ⟨so much has been written pro and ~⟩

⁴con *n* (1589) **1** : an argument or evidence in opposition **2** : the negative position or one holding it ⟨an appraisal of the pros and ~s⟩

⁵con *adj* [by shortening] (1889) : CONFIDENCE

⁶con *vt* **conned; con·ning** (1896) **1** : SWINDLE **2** : PERSUADE, CAJOLE

⁷con *n* [by shortening] (1893) : CONVICT

⁸con *n* [short for *consumption*] *slang* (1915) : a destructive disease of the lungs; *esp* : TUBERCULOSIS

con- — see COM-

con amo·re \ˌkän-ə-'mōr-ē, ˌkō-nə-'mōr-(ˌ)ā, -'môr-\ *adv* [It] (1739) **1** : with love, devotion, or zest **2** : in a tender manner — used as a direction in music

con ani·ma \kä-'nan-ə-ˌmä, kō-'nän-i-\ *adv* [It, lit., with spirit] (1906) : in a spirited manner — used as a direction in music

co·na·tion \kō-'nā-shən\ *n* [L *conation-, conatio* act of attempting, fr. *conatus*, pp. of *conari* to attempt — more at DEACON] (1836) : an inclination (as an instinct, a drive, a wish, or a craving) to act purposefully : IMPULSE 3 — **co·na·tive** \'kō-nət-iv, -ˌnāt-; 'kän-ət-\ *adj*

con brio \kän-'brē-(ˌ)ō, kōn-\ *adv* [It, lit., with vigor] (ca. 1891) : in a vigorous or brisk manner — used as a direction in music

con·ca·nav·a·lin \ˌkän-kə-'nav-ə-lən\ *n* [*com-* + *canavalin* (a noncrystalline globulin found in the jack bean), fr. NL *Canavalia*, genus name of the jack bean] (1919) : either of two crystalline globulins occurring in the jack bean; *esp* : one that is a potent hemagglutinin

¹con·cat·e·nate \kän-'kat-ə-ˌnāt, kən-\ *adj* [ME, fr. LL *concatenatus*, pp. of *concatenare* to link together, fr. L *com-* + *catena* chain — more at CHAIN] (15c) : linked together

²concatenate \-ˌnāt\ *vt* **-nat·ed; -nat·ing** (1598) : to link together in a series or chain — **con·cat·e·na·tion** \(ˌ)kän-ˌkat-ə-'nā-shən, kən-\ *n*

¹con·cave \'kän-'kāv, 'kän-ˌ\ *adj* [MF, fr. L *concavus*, fr. *com-* + *cavus* hollow — more at CAVE] (15c) **1** : hollowed or rounded inward like the inside of a bowl **2** : arched in ; curving in — used of the side of a curve or surface on which neighboring normals to the curve or surface converge and on which lies the chord joining two neighboring points of the curve or surface

²con·cave \'kän-ˌkāv\ *n* (1552) : a concave line or surface

con·cav·i·ty \kän-'kav-ət-ē\ *n, pl* **-ties** (15c) **1** : a concave line, surface, or space ; HOLLOW **2** : the quality or state of being concave

concavo-convex *adj* (1676) **1** : concave on one side and convex on the other **2** : having the concave side curved more than the convex

con·ceal \kən-'sē(ə)l\ *vt* [ME *concelen*, fr. MF *conceler*, fr. L *concelare*, fr. *com-* + *celare* to hide — more at HELL] (14c) **1** : to prevent disclosure or recognition of **2** : to place out of sight *syn* see HIDE — **con·ceal·able** \-'sē-lə-bəl\ *adj* — **con·ceal·er** \-'sē-lər\ *n* — **con·ceal·ing·ly** \-'sē-liŋ-lē\ *adv* — **con·ceal·ment** \-'sē(ə)l-mənt\ *n*

con·cede \kən-'sēd\ *vb* **con·ced·ed; con·ced·ing** [F or L; F *concéder*, fr. L *concedere*, fr. *com-* + *cedere* to yield — more at CEDE] *vt* (1632) **1** : to grant as a right or privilege **2 a** : to accept as true, valid, or accurate ⟨the right of the state to tax is generally *conceded*⟩ **b** : to acknowledge grudgingly or hesitantly ~ *vi* : to make concession ; YIELD *syn* see GRANT — **con·ced·ed·ly** \-'sēd-əd-lē\ *adv* — **con·ced·er** *n*

¹con·ceit \kən-'sēt\ *n* [ME, fr. *conceiven*] (14c) **1 a** (1) : a result of mental activity : THOUGHT (2) : individual opinion **b** : favorable opinion; *esp* : excessive appreciation of one's own worth or virtue **2** : a fancy article **3 a** : a fanciful idea **b** : an elaborate or strained metaphor **c** : use or presence of such conceits in poetry

²conceit *vt* (1557) **1** *obs* : CONCEIVE, UNDERSTAND **2** *dial* : IMAGINE **3** *dial Brit* : to take a fancy to

con·ceit·ed \-'sēt-əd\ *adj* [*¹conceit*] (1593) **1** : ingeniously contrived : FANCIFUL **2** : having an excessively high opinion of oneself — **con·ceit·ed·ly** *adv* — **con·ceit·ed·ness** *n*

con·ceiv·able \kən-'sē-və-bəl\ *adj* (15c) : capable of being conceived : IMAGINABLE — **con·ceiv·abil·i·ty** \kən-ˌsē-və-'bil-ət-ē\ *n* — **con·ceiv·able·ness** \-'sē-və-bəl-nəs\ *n* — **con·ceiv·ably** \-blē\ *adv*

con·ceive \kən-'sēv\ *vb* **con·ceived; con·ceiv·ing** [ME *conceiven*, fr. MF *concevre*, fr. L *concipere* to take in, conceive, fr. *com-* + *capere* to take — more at HEAVE] *vt* (14c) **1 a** : to become pregnant with (young) **b** : to cause to begin : ORIGINATE **2 a** : to take into one's mind ⟨~ a prejudice⟩ **b** : to form a conception of : IMAGINE, IMAGE **3** : to apprehend by reason or imagination : UNDERSTAND **4** : to be of the opinion ~ *vi* **1** : to become pregnant **2** : to have a conception — usu. used with *of* ⟨~s of death as emptiness⟩ *syn* see THINK — **con·ceiv·er** *n*

con·cel·e·brant \kən-'sel-ə-brənt, kän-\ *n* (ca. 1931) : one that concelebrates a Eucharist or Mass

con·cel·e·brate \kən-'sel-ə-ˌbrāt, kän-\ *vb* [L *concelebratus*, pp. of *concelebrare* to celebrate in great numbers, fr. *com-* + *celebrare* to celebrate] *vt* (1879) : to participate in (a Eucharist) as a joint celebrant who recites the canon in unison with other celebrants ~ *vi* : to participate as a celebrant in a concelebrated Eucharist — **con·cel·e·bra·tion** \(ˌ)kän-ˌsel-ə-'brā-shən, kən-\ *n*

con·cent \kən-'sent\ *n* [L *concentus*, fr. *concentus*, pp. of *concinere* to sing together, fr. *com-* + *canere* to sing — more at CHANT] *archaic* (1585) : HARMONY

con·cen·ter \kən-'sent-ər, kän-\ *vb* [F *concentrer*, fr. *com-* + *centre* center] *vt* (1630) : to draw or direct to a common center : CONCENTRATE ~ *vi* : to come to a common center

RCL011396

Case 5:03-cv-04669-JW Document 238-4 Filed 08/27/2004 Page 10 of 11

ag·ri·mo·ny \'ag-rə-ˌmō-nē\ *n, pl* **-nies** [ME, fr. MF & L; MF *aigremoine*, fr. L *agrimonia*, MS var. of *argemonia*, fr. Gk *argemōnē*] (14c) : a common yellow-flowered herb (genus *Agrimonia*) of the rose family having toothed leaves and fruits like burs; *also* : any of several similar or related plants

agro- *comb form* [F, fr. Gk, fr. *agros* field — more at ACRE] 1 : of or belonging to fields or soil : agricultural ⟨*agro*chemical⟩ 2 : agricultural and ⟨*agro*-industrial⟩

ag·ro·chem·i·cal \ag-rō-'kem-i-kəl\ *also* **ag·ri·chem·i·cal** \ag-ri-\ *n* (1956) : an agricultural chemical (as an herbicide or an insecticide)

ag·ro·in·dus·tri·al \ag-rō-in-'dəs-trē-əl\ *adj* (ca. 1940) : of or relating to production (as of power for industry and water for irrigation) for both industrial and agricultural purposes

agron·o·my \ə-'grän-ə-mē\ *n* [prob. fr. F *agronomie*, fr. *agro-* + *-nomie* -nomy] (1814) : a branch of agriculture dealing with field-crop production and soil management — **ag·ro·nom·ic** \ag-rə-'näm-ik\ *also* **ag·ro·nom·i·cal** \ag-rə-'näm-i-kəl\ *adj* — **ag·ro·nom·i·cal·ly** \-i-k(ə-)lē\ *adv* — **agron·o·mist** \ə-'grän-ə-məst\ *n*

aground \ə-'graund\ *adv or adj* (1500) 1 : on or onto the shore or the bottom of a body of water ⟨a ship run ~⟩ 2 : on the ground ⟨planes aloft and ~⟩

ague \'ā-(ˌ)gyü\ *n* [ME, fr. MF *aguë*, fr. ML (*febris*) *acuta*, lit., sharp fever, fr. L, fem. of *acutus* sharp — more at ACUTE] (14c) 1 : a fever (as malaria) marked by paroxysms of chills, fever, and sweating that recur at regular intervals 2 : a fit of shivering : CHILL — **ague·ish** \'ā-gyü-ish\ *adj*

ah \'ä\ *interj* [ME] (13c) — used to express delight, relief, regret, or contempt

aha \ä-'hä\ *interj* [ME] (14c) — used to express surprise, triumph, or derision

ahead \ə-'hed\ *adv or adj* (1596) 1 a : in a forward direction or position : FORWARD b : in front 2 : in, into, or for the future ⟨plan ~⟩ 3 : in or toward a more advantageous position ⟨helped others to get ~⟩ 4 : at or to an earlier time : in advance ⟨make payments ~⟩

ahead of *prep* (1748) 1 : in front or advance of 2 : in excess of

ahem *a throat-clearing sound; often read as* ə-'hem\ *interj* [imit.] (1763) — used esp. to attract attention

ahim·sa \ə-'him-ˌsä\ *n* [Skt *ahiṃsā* noninjury] (1875) : the Hindu and Buddhist doctrine of refraining from harming any living being

ahis·tor·i·cal \ˌā-his-'tȯr-i-kəl, -'tär-\ *or* **a·his·tor·ic** \-ik\ *adj* (1945) : not concerned with or related to history, historical development, or tradition ⟨the ~ attitudes of the radicals⟩

ahold \ə-'hōld\ *n* [prob. fr. the phrase *a hold*] (1872) : HOLD ⟨if you could get ~ of a representative who . . . would come along —Norman Mailer⟩

A·ho·ra·zon \'ä-hə-ˌrāz-ᵊn\ *n* (1936) : the uppermost dark-colored layer of a soil consisting largely of partly disintegrated organic debris

ahoy \ə-'hȯi\ *interj* [*a-* (as in *aha*) + *hoy*] (1751) — used in hailing ⟨ship ~⟩

Ah·ri·man \'är-i-mən, -ˌman\ *n* [Per, modif. of Av *aŋrō mainyuš* hostile spirit] : Ahura Mazda's antagonist who is a spirit of darkness and evil in Zoroastrianism

Ahu·ra Maz·da \ə-ˌhür-ə-'maz-də, ä-ˌhür-\ *n* [Av *Ahuramazda*, lit., wise god] : the Supreme Being represented as a deity of goodness and light in Zoroastrianism

ai \'ī, ä-'ē\ *n* [Pg *ai*, or Sp *ai*, fr. Tupi *ai*] (1693) : a sloth (genus *Bradypus*) with three claws on each front foot

Ai·as \'ī-əs\ *n* [Gk] : AJAX

ai·blins \'ā-blənz\ *adv* [*able* + *-lings*, *-lins* -lings] *chiefly Scot* (1597) : PERHAPS

aid \'ād\ *vb* [ME *eyden*, fr. MF *aider*, fr. L *adjutare*, fr. *adjutus*, pp. of *adjuvare*, fr. *ad-* + *juvare* to help] *vt* (15c) : to provide with what is useful or necessary in achieving an end ~ *vi* : to give assistance *syn* see HELP — **aid·er** *n*

²aid *n* (15c) 1 : a subsidy granted to the king by the English parliament until the 18th century for an extraordinary purpose 2 a : the act of helping b : help given : ASSISTANCE; *specif* : tangible means of assistance (as money or supplies) 3 a : an assisting person or group — compare AIDE b : something by which assistance is given ; an assisting device ⟨an ~ to understanding⟩ ⟨a visual ~⟩; *specif* : HEARING AID 4 : a tribute paid by a vassal to his lord

aide \'ād\ *n* [short for *aide-de-camp*] (1777) : a person who acts as an assistant; *specif* : a military officer acting as assistant to a superior

aide–de–camp \ˌād-di-'kamp, -'käⁿ\ *n, pl* **aides–de–camp** \ˌād(z)-di-\ [F *aide de camp*, lit., camp assistant] (1670) : a military aide; *also* : a civilian aide usu. to an executive

aide–mé·moire \ˌād-mām-'wär\ *n, pl* **aide–mémoire** [F, fr. *aider* to aid + *mémoire* memory] (1846) 1 : an aid to the memory; *esp* : a mnemonic device 2 : a written summary or outline of important items of a proposed agreement or diplomatic communication : MEMORANDUM

aid·man \'ād-ˌman\ *n* (1944) : an army medical corpsman attached to a field unit

AIDS \'ādz\ *n* [acquired immunodeficiency syndrome] (1982) : a condition of acquired immunological deficiency associated esp. with male homosexuality and intravenous drug abuse

ai·grette \ā-'gret, 'ā-ˌ\ *n* [F, plume, egret, fr. MF — more at EGRET] (1630) 1 : a spray of feathers (as of the egret) for the head 2 : a spray of gems worn on a hat or in the hair

ai·guille \ā-'gwē(ə)l, -'gwē\ *n* [F, lit., needle — more at AGLET] (1816) 1 : a sharp-pointed pinnacle of rock 2 : an instrument for boring holes in stone or other masonry materials

ai·guil·lette \ˌā-gwi-'let\ *n* [F — more at AGLET] (1816) : AGLET; *specif* : a shoulder cord worn by designated military aides — compare FOURRAGÈRE

ai·ki·do \ˌī-ki-'dō, ˌī-kē-(ˌ)dō\ *n* [Jp *aikidō*, fr. *ai-* match, coordinate + *ki* breath, spirit + *dō* art, way] (1963) : a Japanese art of self-defense employing locks and holds and utilizing the principle of nonresistance to cause an opponent's own momentum to work against him

¹ail \'ā(ə)l\ *vb* [ME *eilen*, fr. OE *eglan*; akin to MLG *egelen* to annoy] *vt* (bef. 12c) : to give physical or emotional pain, discomfort, or trouble to ~ *vi* : to have something the matter; *esp* : to suffer ill health

²ail *n* (13c) : AILMENT

ai·lan·thus \ā-'lan(t)-thəs\ *n* [NL, fr. Amboinese *ai lanto*, lit., tree (of) heaven] (ca. 1807) : any of a small Asian genus (*Ailanthus* of the family Simaroubaceae, the ailanthus family) of chiefly tropical trees and shrubs with bitter bark, pinnate leaves, and terminal panicles of ill-scented greenish flowers

ai·le·ron \'ā-lə-ˌrän\ *n* [F, fr. dim. of *aile* wing — more at AISLE] (1909) : a movable part of an airplane wing or a movable airfoil external to the wing at the trailing edge for imparting a rolling motion and thus providing lateral control —see AIRPLANE illustration

ail·ment \'ā(ə)l-mənt\ *n* (ca. 1706) 1 : a bodily disorder or chronic disease 2 : UNREST, UNEASINESS

ai·lu·ro·phile \ī-'lůr-ə-ˌfīl, ā-\ *n* [Gk *ailouros* cat] (1927) : a cat fancier : a lover of cats

ai·lu·ro·phobe \-ˌfōb\ *n* (1905) : one who hates or fears cats

¹aim \'ām\ *vb* [ME *aimen*, fr. MF *aesmer & esmer*; MF *aesmer*, fr. OF, fr. *a-* (fr. L *ad-*) + *esmer* to estimate, fr. L *aestimare* — more at ESTEEM] *vi* (14c) 1 : to direct a course; *specif* : to point a weapon at an object 2 : ASPIRE, INTEND ⟨~*s* to reform the government⟩ ~ *vt* 1 *obs* : GUESS, CONJECTURE 2 a : POINT b : to direct to or toward a specified object or goal ⟨a program ~*ed* at reducing pollution⟩

²aim *n* (14c) 1 *obs* : MARK, TARGET 2 a : the pointing of a weapon at a mark b : the ability to hit a target ⟨his ~ was deadly⟩ c : a weapon's accuracy or effectiveness 3 *obs* a : CONJECTURE, GUESS b : the directing of effort toward a goal 4 : a clearly directed intent or purpose *syn* see INTENTION — **aim·less** \-ləs\ *adj* — **aim·less·ly** *adv* — **aim·less·ness** *n*

ain \'ān\ *adj* [prob. fr. ON *eiginn*] *Scot* (bef. 12c) : OWN

ain't \'ānt\ [prob. contr. of *are not*] 1 a : are not b : is not c : am not 2 *substand* a : have not b : has not

usage Although disapproved by many and more common in less educated speech, *ain't* is used orally in most parts of the U.S. Its use by educated people is in sense 1 esp. orally in the phrase *ain't I*. At all levels of education it is used deliberately to catch attention or for emphasis both in speech ⟨makes me look half-witted, which I *ain't* — Sir Winston Churchill⟩ and in writing ⟨the wackiness of movies, once so deliciously amusing, *ain't* funny anymore —Richard Schickel⟩ and is found frequently in a few fixed phrases and constructions ⟨leftovers *ain't* what they used to be —*Apartment Life*⟩ ⟨well — class it *ain't* — Cleveland Amory⟩ ⟨and that *ain't* hay⟩ ⟨you *ain't* seen nothin' yet⟩ It is also used for metrical reasons in popular songs ⟨It *Ain't* Necessarily So⟩ ⟨*Ain't* She Sweet⟩ ⟨the old gray mare, she *ain't* what she used to be⟩

Ai·nu \'ī-(ˌ)nü\ *n, pl* **Ainu** *or* **Ainus** [Ainu, lit., man] (1819) 1 : a member of an indigenous Caucasoid people of Japan 2 : the language of the Ainu people

ai·o·li \(ˌ)ī-'ō-lē, (ˌ)ä-\ *n* [Prov, fr. *ai* garlic + *oli* oil—more at OIL] (ca. 1900) : a sauce made of crushed garlic, egg yolks, olive oil, and lemon juice and sometimes potato : garlic mayonnaise

¹air \'a(ə)r, 'e(ə)r\ *n, often attrib* [ME, fr. MF, fr. L *aer*, fr. Gk *aēr*] (14c) 1 a *archaic* : BREATH b : the mixture of invisible odorless tasteless gases (as nitrogen and oxygen) that surrounds the earth c : a light breeze 2 a : empty space b : NOTHINGNESS ⟨vanished into thin ~⟩ c : a sudden severance of relations ⟨she gave him the ~⟩ 3 : COMPRESSED AIR 4 a (1) : AIRCRAFT ⟨go by ~⟩ (2) : AVIATION ⟨~ safety⟩ ⟨~ rights⟩ (3) : AIR FORCE ⟨~ headquarters⟩ b : the medium of transmission of radio waves; *also* : RADIO, TELEVISION ⟨went on the ~⟩ 5 : public utterance ⟨he gave ~ to his opinion⟩ 6 a : the look, appearance, or bearing of a person esp. as expressive of some personal quality or emotion : DEMEANOR ⟨an ~ of dignity⟩ b : an artificial or affected manner : HAUGHTINESS ⟨to put on ~*s*⟩ c : outward appearance of a thing ⟨an ~ of luxury⟩ d : a surrounding or pervading influence : ATMOSPHERE ⟨an ~ of mystery⟩ 7 [prob. trans. of It *aria*] *Elizabethan & Jacobean music* : an accompanied song or melody usu. strophic form b : the chief voice part or melody in choral music c : TUNE, MELODY 8 : a football offense utilizing primarily the forward pass ⟨behind by three touchdowns and forced to take to the ~⟩ 9 : air-conditioning system *syn* see POSE — **air·less** \-ləs\ *adj* — **air·less·ness** *n* — **in the air** : in wide circulation : ABOUT — **up in the air** : not yet settled

²air *vt* (1530) 1 : to expose to the air for drying, purifying, or refreshing : VENTILATE — often used with *out* 2 : to expose to public view or bring to public notice 3 : to transmit by radio or television ⟨~ a program⟩ ~ *vi* 1 : to become exposed to the open air 2 : to become broadcast ⟨the program ~*s* daily⟩ *syn* see EXPRESS

air bag *n* (1969) : an automatically inflating bag in front of riders in an automobile to protect them from pitching forward into solid parts in case of an accident

air base *n* (1915) : a base of operations for military aircraft

air bladder *n* (1731) : a sac containing gas and esp. air; *esp* : a hydrostatic organ present in most fishes that serves as an accessory respiratory organ

air·boat \'a(ə)r-ˌbōt, 'e(ə)r-\ *n* (1946) : a shallow-draft boat driven by an airplane propeller and steered by an airplane rudder

air·borne \-ˌbō(ə)rn, -ˌbó(ə)rn\ *adj* (1641) 1 : supported wholly by aerodynamic and aerostatic forces 2 : transported by air

air brake *n* (1871) 1 : a brake operated by a piston driven by compressed air 2 : a surface (as an aileron) that may be projected into the airstream for lowering the speed of an airplane

¹air·brush \-ˌbrəsh\ *n* (ca. 1889) : an atomizer for applying by compressed air a fine spray (as of paint or liquid color)

²airbrush *vt* (1938) : to paint, treat, or alter with an airbrush

air·burst \-ˌbərst\ *n* (1917) : the burst of a shell or bomb in the air

air·bus \-ˌbəs\ *n* (1945) : a short-range or medium-range subsonic passenger airplane

air chief marshal *n* (ca. 1919) : a commissioned officer in the British air force who ranks with a general in the army

air coach *n* (1948) : a passenger airliner offering service at less than first-class rates usu. with curtailed accommodations

air commodore *n* (ca. 1919) : a commissioned officer in the British air force who ranks with a brigadier in the army

air–con·di·tion \ˌa(ə)r-kən-'dish-ən, ˌe(ə)r-\ *vt* [back-formation fr. *conditioning*] (1933) : to equip (as a building) with an apparatus washing air and controlling its humidity and temperature; *also* : to subject (air) to these processes — **air con·di·tion·er** \-ᵊnər\ *n* — **air–con·di·tion·ing** \-'dish-(ə-)niŋ\ *n*

air–cool \'a(ə)
ing] (1899) :
by air withou
air-craft \'a(ə,
weight-carryi
either by its o
its surfaces
aircraft carrie
planes can be
air-crew \'a(ə)r
air–cushion vel
air-date \-ˌdāt\
air-drome \'a(ə,
air-drop \-ˌdräp
chute from a
\-ˌdräp-ə-bəl\
air-dry \-ˌdrī\
moisture is giv
Aire·dale terrie
river, England
wiry, black-an
airer \'a(ə)r-ər,
aired or dried
Air Express *ser*
air-fare \'a(ə)r-ˌ
plane
air-field \-ˌfēld\
PORT
air-flow \-ˌflō\ *n*
around parts o
immersed in it
air-foil \-ˌfȯil\ *n*
blade) designe
relative to the s
air force *n* (1917
fare 2 : a unit
than a comman
air-frame \-ˌfrām
craft, rocket veh
air-freight \-ˌfrāt
: the charge for
air-glow \-ˌglō\
night, that origi
and that is asso
solar radiation
air gun *n* (ca. 17
compressed air
pressed air; *esp*
air-head \-ˌhed\
in hostile territo
bringing in troop
air hole *n* (15c)
frozen over in ice
air-ing \'a(ə)r-iŋ,
or freshening 2
mote health or
radio or televisio
air lane *n* (ca. 191
air letter *n* (1920)
that can be folde
outside
air-lift \'a(ə)r-ˌlift,
passengers by air
airlift *vt*
air-line \-ˌlīn\ *n* (
equipment, route
air line *n* (1813) :
air-lin-er \-ˌlī-nər\
air lock *n* (1857)
and the working
mediate chambe
temperature 2 :
liquid ought to cir
air-mail \'a(ə)r-ˌmā
ing mail by aircra
air-man \-mən\ *n*
aviation technicia
enlisted man of
man ranking abov
airman basic *n* (
force
airman first class
above an airman
air-man-ship \'a(ə)r
navigating airplan
air marshal *n* (191
who ranks with a
air mass *n* (1893)
miles horizontally
taining its it trave
humidity at any giv
air mattress *n* (19
Air Medal *n* (194
ous achievement
air mile *n* (1919)
feet
air–mind·ed \'a(ə)
or in air travel
air-mo·bile \-mō
tro, or being a unit
areas used by he
air park \-ˌpärk\
air piracy *n* (1948)

RCL011397

\'des-kant\ *n* [ME *dyscant*, fr. ONF & ML; ONF *descant*, fr. ML *discantus*, fr. L *dis-* + *cantus* song — more at CHANT] (14c) **1 a** : the counterpoint sung above the plainsong of the tenor **b** : the art of composing or improvising contrapuntal part music; *also* : music so composed or improvised **c** : SOPRANO, TREBLE **d** : a composed counterpoint to a simple melody sung typically by some of the sopranos **2** : discourse or comment on a theme

de-scant \,des-'kant, ,dēs-, ,dis-\ *vi* (15c) **1** : to sing or play a descant; SING **2** : COMMENT, DISCOURSE

de-scend \di-'send\ *vb* [ME *descenden*, fr. OF *descendre*, fr. L *descendere*, fr. *de-* + *scandere* to climb — more at SCAN] *vi* (14c) **1** : to pass from a higher place or level to a lower one ⟨~ed from the platform⟩ **2** : to pass from discussion from what is logically prior or more comprehensive **3** : to come down from a stock or source : DERIVE — usu. used figuratively ⟨was ~ed from an ancient family⟩ **b** : to pass by inheritance (an heirloom that has ~ed in the family) **c** : to pass by tradition (songs ~ed from early ballads) **4** : to incline, lead, or move downward ⟨the road ~s to the river⟩ **5** : to swoop or pounce : make a sudden attack ⟨the plague ~ed upon them⟩ **6** : to proceed in a sequence or gradation from higher to lower or from more remote to nearer or more recent **7 a** : to lower oneself in status or quality : stoop **b** : to worsen and sink in condition or estimation ~ *vt* **1** : to pass, move, or climb down or down along **2** : to extend down along

de-scend-ible \-'sen-də-bəl\ *adj*

de-scend-ant \di-'sen-dənt\ *adj* [MF & L; MF *descendant*, fr. L *descendent-*, *descendens*, prp. of *descendere*] (1572) **1** : moving or directed downward **2** : proceeding from an ancestor or source

descendant or **descendent** *n* [F & L; F *descendant*, fr. LL *descendent-*, *descendens*, fr. L] (1600) **1** : one descended from another or from a common stock **2** : one deriving directly from a precursor or prototype

descender \di-'sen-dər, 'dē-\ *n* (1802) : the part of a lowercase letter (as p) that descends below the main body of the letter; *also* : a letter having such a part

de-scen-sion \di-'sen-chən\ *n*, *archaic* (14c) : DESCENT 2

de-scent \di-'sent\ *n* [ME, fr. MF *descente*, fr. OF *descendre*] (14c) **1 a** : derivation from an ancestor : BIRTH, LINEAGE ⟨of French ~⟩ **b** : orientation or devolution of an estate by inheritance usu. in the descending line **c** : the fact or process of originating from an ancestral stock **d** : the shaping or development in nature and character by transmission from a source : DERIVATION **2** : the act or process of descending **3** : a step downward in a scale of gradation; *specif* : one generation in an ancestral line or genealogical scale **4 a** : an inclination downward : SLOPE **b** : a descending way (as a downgrade or stairway) **c** *obs* : the lowest part **5 a** : ATTACK, INVASION **b** : a sudden disconcerting appearance (as for a visit) **6** : a downward step (as in station or value) : DECLINE ⟨~ of the family to actual poverty⟩

de-scribe \di-'skrīb\ *vt* **de-scribed; de-scrib-ing** [L *describere*, fr. *de-* + *scribere* to write — more at SCRIBE] (15c) **1** : to represent or give an account of in words ⟨~ a picture⟩ **2** : to represent by a figure, model, or picture : DELINEATE **3** *obs* : DISTRIBUTE **4** : to trace or traverse the outline of ⟨~ a circle⟩ **5** *archaic* : OBSERVE, PERCEIVE — **de-scrib-able** \-'skrī-bə-bəl\ *adj* — **de-scrib-er** *n*

de-scrip-tion \di-'skrip-shən\ *n* [ME *descripcioun*, fr. MF & L; MF *description*, fr. L *description-*, *descriptio*, fr. *descriptus*, pp. of *describere*] (14c) **1 a** : an act of describing; *specif* : discourse intended to give a mental image of something experienced (as a scene, person, or sensation) **b** : a descriptive statement or account ⟨a fascinating ~ of his adventures⟩ **2** : kind or character esp. as determined by salient features (opposed to any tax of so radical a ~) *syn* see TYPE

de-scrip-tive \di-'skrip-tiv\ *adj* (1751) **1** : serving to describe ⟨a ~ account⟩ **2** : referring to, constituting, or grounded in matters of observation or experience ⟨the ~ basis of science⟩ **3** *of a modifier* **a** : expressing the quality, kind, or condition of what is denoted by the modified term ⟨*hot* in "hot water" is a ~ adjective⟩ **b** : NONRESTRICTIVE **4** : of, relating to, or dealing with the structure of a language at a particular time usu. with exclusion of historical and comparative data ⟨~ linguistics⟩ — **de-scrip-tive-ly** *adv* — **de-scrip-tive-ness** *n*

de-scrip-tor \di-'skrip-tər\ *n* (1951) : something (as an index term) used to identify an item (as a subject or document) esp. in an information retrieval system

de-scry \di-'skrī\ *vt* **de-scried; de-scry-ing** [ME *descrien*, fr. MF *descrier* to proclaim, decry] (14c) **1 a** : to catch sight of **b** : to find out : DISCOVER **2** *obs* : to make known : REVEAL

descry *n*, *obs* (1605) : discovery or view from afar

Des-demo-na \,dez-də-'mō-nə\ *n* : the wife of Othello in Shakespeare's *Othello*

des-e-crate \'des-i-,krāt\ *vt* **-crat-ed; -crat-ing** [*de-* + *-secrate* (as in *consecrate*)] (1677) **1** : to violate the sanctity of : PROFANE **2** : to treat irreverently or contemptuously often in a way that provokes outrage on the part of others ⟨the kind of shore development . . . that has *desecrated* so many waterfronts — John Fischer⟩ — **des-e-crat-er** *or* **des-e-cra-tor** \-,krāt-ər\ *n*

des-e-cra-tion \,des-i-'krā-shən\ *n* (1717) : an act or instance of desecrating; the state of being desecrated

de-segre-gate \(,)dē-'seg-ri-,gāt\ *vt* (1952) : to eliminate segregation in; *specif* : to free of any law, provision, or practice requiring isolation of the members of a particular race in separate units ~ *vi* : to bring about desegregation

de-segre-ga-tion \(,)dē-,seg-ri-'gā-shən\ *n* (1951) **1** : the act or process or an instance of desegregating **2** : the state of being desegregated

de-select \,dē-sə-'lekt\ *vt* (1965) : to dismiss (a trainee) from a training program

de-sen-si-tize \(,)dē-'sen(t)-sə-,tīz\ *vt* (1898) **1** : to make (a sensitized or hypersensitive individual) insensitive or nonreactive to a sensitizing agent **2** : to make (a photographic material) less sensitive or completely insensitive to radiation **3** : to make emotionally insensitive or callous; *specif* : to extinguish an emotional response (as of fear, anxiety, or guilt) to stimuli that formerly induced it — **de-sen-si-ti-za-tion** \(,)dē-,sen-sət-ə-'zā-shən, -,sen-stə-'zī-\ *n* — **de-sen-si-tiz-er** \(,)dē-'sen-sə-,tī-zər\ *n*

desert \'dez-ərt\ *n* [ME, fr. OF, fr. LL *desertum*, fr. L, neut. of *desertus*, pp. of *deserere* to desert, fr. *de-* + *serere* to join together — more at SERIES] (13c) **1 a** : arid barren land; *esp* : a tract incapable of supporting any considerable population without an artificial water supply **b** : an area of water apparently devoid of life **2** *archaic* : a wild uninhabited and uncultivated tract **3** : a desolate or forbidding area ⟨lost in a ~ of doubt⟩ — **de-ser-tic** \də-'zərt-ik\ *adj*

²de-sert \'dez-ərt\ *adj* (13c) **1** : desolate and sparsely occupied or unoccupied ⟨a ~ island⟩ **2** : of or relating to a desert **3** *archaic* : FORSAKEN

³de-sert \di-'zərt\ *n* [ME *deserte*, fr. OF, fr. fem. of *desert*, pp. of *deservir* to deserve] (13c) **1** : the quality or fact of deserving reward or punishment **2** : deserved reward or punishment — usu. used in plural ⟨got his just ~s⟩ **3** : EXCELLENCE, WORTH

⁴de-sert \di-'zərt\ *vb* [F *déserter*, fr. LL *desertare*, fr. *desertus*] *vt* (1603) **1** : to withdraw from or leave usu. without intent to return **2 a** : to leave in the lurch ⟨~ a friend in trouble⟩ **b** : to abandon (military service) without leave ~ *vi* : to quit one's post, allegiance, or service without leave or justification; *esp* : to absent oneself from military duty without leave and without intent to return *syn* see ABANDON — **de-sert-er** *n*

de-sert-i-fi-ca-tion \di-,zərt-ə-fə-'kā-shən\ *n* [*desert* + *-ification* (as in *saponification*)] (1974) : the process of becoming arid land or desert (as from land mismanagement or climate change)

de-ser-tion \di-'zər-shən\ *n* (1591) **1** : an act of deserting; *esp* : the abandonment without consent or legal justification of a person, post, or relationship and the associated duties and obligations ⟨sued for divorce on grounds of ~⟩ **2** : a state of being deserted or forsaken

desert locust *n* (1944) : a destructive migratory locust (*Schistocerca gregaria*) of southwestern Asia and parts of northern Africa

desert soil *n* (ca. 1938) : a soil that develops under sparse shrub vegetation in warm to cool arid climates with a light-colored surface soil usu. underlain by calcareous material and a hardpan layer

de-serve \di-'zərv\ *vb* **de-served; de-serv-ing** [ME *deserven*, fr. OF *deservir*, fr. L *deservire* to serve zealously, fr. *de-* + *servire* to serve] *vt* (13c) : to be worthy of : MERIT ⟨~s another chance⟩ ~ *vi* : to be worthy, fit, or suitable for some reward or requital ⟨have become recognized as they ~ —T. S. Eliot⟩ — **de-serv-er** *n*

de-served \-'zərvd\ *adj* (ca. 1582) : of, relating to, or being that which one deserves ⟨a ~ reputation⟩ — **de-served-ly** \-'zər-vəd-lē, -'zərv-dlē\ *adv* — **de-served-ness** \-'zər-vəd-nəs, -'zərv(d)-nəs\ *n*

¹de-serv-ing \-'zər-viŋ\ *n* (14c) : DESERT, MERIT ⟨reward the proud according to their ~s —Charles Kingsley⟩

²deserving *adj* (1576) : MERITORIOUS, WORTHY; *specif* : meriting financial aid ⟨scholarships for ~ students⟩

de-sex \(')dē-'seks\ *vt* (1911) : CASTRATE, SPAY

de-sex-u-al-ize \(')dē-'sekh-(ə-)wə-,līz, -'sek-shə-,līz\ *vt* (1894) **1** : to deprive of sexual characters or power **2** : to divest of sexual quality — **de-sex-u-al-iza-tion** \(,)dē-,sekh-(ə-)wə-lə-'zā-shən, -'sek-shə-lə-\ *n*

des-ha-bille \,des-ə-'bā(ə)l, -'bil, -'bē\ *var of* DISHABILLE

des-ic-cant \'des-i-kənt\ *n* (1676) : a drying agent (as calcium chloride)

des-ic-cate \'des-i-,kāt\ *vb* **-cat-ed; -cat-ing** [L *desiccatus*, pp. of *desiccare* to dry up, fr. *de-* + *siccare* to dry, fr. *siccus* dry — more at SACK] *vt* (1575) **1** : to dry up **2** : to preserve (a food) by drying : DEHYDRATE **3** : to drain of emotional or intellectual vitality ~ *vi* : to become dried up — **des-ic-ca-tion** \,des-i-'kā-shən\ *n* — **des-ic-ca-tive** \'des-i-,kāt-iv, di-'sik-ət-\ *adj* — **des-ic-ca-tor** \'des-i-,kāt-ər\ *n*

de-sid-er-ate \di-'sid-ə-,rāt, -'zid-\ *vt* **-at-ed; -at-ing** [L *desideratus*, pp. of *desiderare* to desire] (1645) : to entertain or express a wish to have or attain — **de-sid-er-a-tion** \-,sid-ə-'rā-shən, -,zid-\ *n* — **de-sid-er-a-tive** \-'sid-ə-,rāt-iv, -'sid-(ə-)rət-, -'zid-\ *adj*

de-sid-er-a-tum \-,sid-ə-'rāt-əm, -,zid-, -'rāt-\ *n, pl* **-ta** \-ə\ [L, neut. of *desideratus*] (1652) : something desired as essential

¹de-sign \di-'zīn\ *vb* [MF *designer*, fr. L *designare*, fr. *de-* + *signare* to mark, mark out — more at SIGN] *vt* (1548) **1 a** : to conceive and plan out in the mind ⟨he ~ed to excel in his studies⟩ **b** : to have as a purpose : INTEND ⟨he ~ed to excel in his studies⟩ **c** : to devise for a specific function or end ⟨a book ~ed primarily as a college textbook⟩ **2** *archaic* : to indicate with a distinctive mark, sign, or name **3 a** : to make a drawing, pattern, or sketch of **b** : to draw the plans for **c** : to create, fashion, execute, or construct according to plan : DEVISE, CONTRIVE ~ *vi* **1** : to conceive or execute a plan **2** : to draw, lay out, or prepare a design — **de-sign-ed-ly** \-'zī-nəd-lē\ *adv*

²design *n* (1588) **1 a** : a particular purpose held in view by an individual or group ⟨he has ambitious ~s for his son⟩ **b** : deliberate purposive planning ⟨battle was joined . . . more by accident than ~ —John Buchan⟩ **2** : a mental project or scheme in which means to an end are laid down **3 a** : a deliberate undercover project or scheme; PLOT **b** *pl* : aggressive or evil intent — used with on or *against* ⟨he has ~s on the money⟩ **4** : a preliminary sketch or outline showing the main features of something to be executed : DELINEATION **5 a** : an underlying scheme that governs functioning, developing, or unfolding : PATTERN, MOTIF ⟨the general ~ of the epic⟩ **b** : a plan or protocol for carrying out or accomplishing something (esp. a scientific experiment); *also* : the process of preparing this **6** : the arrangement of elements or details in a product or work of art **7** : a decorative pattern **8** : the creative art of executing aesthetic or functional designs *syn* see INTENTION, PLAN

¹des-ig-nate \'dez-ig-,nāt, -nət\ *adj* [L *designatus*, pp. of *designare*] (1646) : chosen for an office but not yet installed ⟨ambassador ~⟩

²des-ig-nate \-,nāt\ *vt* **-nat-ed; -nat-ing** (1791) **1** : to indicate and set apart for a specific purpose, office, or duty **2 a** : to point out the location of ⟨a marker *designating* the crest of the floodwaters⟩ **b** : INDICATE ⟨any task *designated* by the employer⟩ **c** : to distinguish as to class ⟨the area we ~ as that of spiritual values —J. B. Conant⟩ **d** : SPECIFY, STIPULATE **3** : DENOTE **4** : to call by a distinctive title, term,

RCL011398

EXHIBIT 4
(Part 2 of 2)

con-clu-sion \kən-'klü-zhən\ *n* [ME, fr. MF, fr. L *conclusion-, conclusio,* fr. *conclusus* pp. of *concludere*] (14c) 1 **a** : a reasoned judgment : INFERENCE **b** : the necessary consequence of two or more propositions taken as premises; *esp* : the inferred proposition of a syllogism 2 : the last part of something: as **a** : RESULT, OUTCOME **b** *pl* : trial of strength or skill — used in the phrase *try conclusions* **c** : a final summation **d** : the final decision in a law case **e** : the final part of a pleading in law 3 : an act or instance of concluding

con-clu-sive \-'klü-siv, -ziv\ *adj* (1612) 1 : of or relating to a conclusion 2 : putting an end to debate or question esp. by reason of irrefutability — **con-clu-sive-ly** *adv* — **con-clu-sive-ness** *n*

SYN CONCLUSIVE, DECISIVE, DETERMINATIVE, DEFINITIVE mean bringing to an end. CONCLUSIVE applies to reasoning or logical proof that puts an end to debate or questioning; DECISIVE may apply to something that ends a controversy; a contest; or any uncertainty; DETERMINATIVE adds an implication of giving a fixed course or direction; DEFINITIVE applies to what is put forth as final and permanent

con-coct \kən-'käkt, kän-\ *vt* [L *concoctus,* pp. of *concoquere* to cook together, fr. *com-* + *coquere* to cook — more at COOK] (1675) 1 : to prepare by combining crude materials 2 : DEVISE, FABRICATE — **con-coct-er** *n* — **con-coc-tion** \-'käk-shən\ *n* — **con-coc-tive** \-'käk-tiv\ *adj*

con-com-i-tance \kən-'käm-ət-ən(t)s, kän-\ *n* (1535) : ACCOMPANIMENT; *esp* : a conjunction that is regular and is marked by correlative variation of accompanying elements

con-com-i-tant \-ət-ənt\ *adj* [L *concomitant-, concomitans,* prp. of *concomitari* to accompany, fr. *com-* + *comitari* to accompany, fr. *comit-, comes* companion — more at COUNT] (1607) : accompanying esp. in a subordinate or incidental way — **con-com-i-tant-ly** *adv*

concomitant *n* (1621) : something that accompanies or is collaterally connected with something else : ACCOMPANIMENT

con-cord \'kän-,kȯ(ə)rd, 'kaŋ-\ *n* [ME, fr. MF *concorde,* fr. L *concordia,* fr. *concord-, concors* agreeing, fr. *com-* + *cord-, cor* heart — more at HEART] (14c) 1 **a** : a state of agreement : HARMONY **b** : a simultaneous occurrence of two or more musical tones that produces an impression of agreeableness or resolution on a listener — compare DISCORD 2 : agreement by stipulation, compact, or covenant 3 : grammatical agreement

con-cor-dance \kən-'kȯrd-ən(t)s, kän-\ *n* [ME, fr. MF, fr. ML *concordantia,* fr. L *concordant-, concordans* prp. of *concordare* to agree, fr. *concord-, concors*] (14c) 1 : an alphabetical index of the principal words in a book or the works of an author with their immediate context 2 : CONCORD, AGREEMENT

con-cor-dant \-ənt\ *adj* [ME, fr. MF, fr. L *concordant-, concordans*] (15c) : CONSONANT, AGREEING — **con-cor-dant-ly** *adv*

con-cor-dat \kən-'kȯr-,dat\ *n* [F, fr. ML *concordatum,* fr. L, neut. of *concordatus,* pp. of *concordare*] (1616) : COMPACT, COVENANT; *specif* : an agreement between a pope and a sovereign or government for the regulation of ecclesiastical matters

con-cours d'e-le-gance \(,)kō⁼-,kú(ə)r-,dā-lē-'gäⁿs\ *n* [F *concours d'élégance,* lit., competition of elegance] (1950) : a show or contest of vehicles and accessories in which the entries are judged chiefly on excellence of appearance and turnout

con-course \'kän-,kȯ(ə)rs, 'käŋ-, -,kȯ(ə)rs\ *n* [ME, fr. MF & L; MF *concours,* fr. L *concursus,* fr. *concursus,* pp. of *concurrere* to run together — more at CONCUR] (14c) 1 : an act or process of coming together and merging 2 : a meeting produced by voluntary or spontaneous coming together 3 **a** : an open space where roads or paths meet **b** : an open space or hall (as in a railroad terminal) where crowds gather

con-cres-cence \kən-'kres-ᵊn(t)s, kän-\ *n* [L *concrescentia,* fr. *concrescent-, concrescens,* prp. of *concrescere* to grow together, fr. *com-* + *crescere* to grow — more at CRESCENT] (1614) 1 : increase by the addition of particles 2 : a growing together : COALESCENCE; *esp* : convergence and fusion of the lateral lips of the blastopore to form the primordium of an embryo — **con-cres-cent** \-ᵊnt\ *adj*

con-crete \'kän-,krēt, kän-, kȯn-\ *adj* [ME, fr. L *concretus,* fr. pp. of *concrescere*] (14c) 1 : naming a real thing or class of things (the word *poem* is ~, *poetry* is abstract) 2 : formed by coalition of particles into one solid mass 3 **a** : characterized by or belonging to immediate experience of actual things or events **b** : SPECIFIC, PARTICULAR **c** : REAL, TANGIBLE 4 : relating to or made of concrete — **con-crete-ly** *adv* — **con-crete-ness** *n*

concrete \'kän-,krēt, kän-\ *vb* **con-cret-ed; con-cret-ing** *vt* (1635) 1 **a** : to form into a solid mass : SOLIDIFY **b** : COMBINE, BLEND 2 : to make actual or real : cause to take on the qualities of reality 3 : to cover with, form of, or set in concrete ~ *vi* : to become concreted

con-crete \'kän-,krēt, (')kän-\ *n* (1655) 1 : a mass formed by concretion or coalescence of separate particles of matter in one body 2 : a hard strong building material made by mixing a cementing material (as portland cement) and a mineral aggregate (as sand and gravel) with sufficient water to cause the cement to set and bind the entire mass 3 : a waxy essence of flowers prepared by extraction and evaporation and used in perfumery

concrete music *n* (1950) : MUSIQUE CONCRÈTE

concrete poetry *n* (1958) : poetry in which the poet's intent is conveyed by the graphic patterns of letters, words, or symbols rather than by the conventional arrangement of words

con-cre-tion \kän-'krē-shən, kən-\ *n* (1514) 1 : something concreted: as **a** : a hard usu. inorganic mass (as a bezoar or tophus) formed in a living body **b** : a mass of mineral matter found generally in rock of a composition different from its own and produced by deposition from aqueous solution in the rock 2 : the act or process of concreting : the state of being concreted (~ of ideas in an hypothesis) — **con-cre-tion-ar-y** \-shə-,ner-ē\ *adj*

concretism \'kän-'krēt-,iz-əm, 'kän-\ *n* (1865) : representation of abstractions in concrete *esp* : the theory or practice of concrete poetry

concretist \-'krēt-əst, -,krēt-\ *n*

concretize \-'krē-,tīz\ *vb* **-tized; -tiz-ing** *vt* (1884) : to make concrete, specific, or definite (tried to ~ his ideas) ~ *vi* : to become concrete — **con-cret-i-za-tion** \(,)kän-,krēt-ə-'zā-shən\ *n*

con-cu-bi-nage \'kän-kyü-bə-nij, kən-\ *n* (14c) 1 : cohabitation of persons not legally married 2 : the state of being a concubine

con-cu-bine \'kän-kyü-,bīn, 'kän-\ *n* [ME, fr. MF, fr. L *concubina,* fr. *concubare* to lie — more at HIP] (14c) 1 : a woman living in a recognized state of concubinage 2 : MISTRESS

con-cu-pis-cence \kän-'kyü-pə-sən(t)s, kən-\ *n* [ME, fr. MF, fr. LL *concupiscentia,* fr. L *concupiscent-, concupiscens,* prp. of *concupiscere* to desire ardently, fr. *com-* + *cupere* to desire — more at COVET] (14c) : strong desire; *esp* : sexual desire — **con-cu-pis-cent** \-sənt\ *adj*

con-cu-pis-ci-ble \-'kyü-pə-sə-bəl\ *adj* [ME, fr. MF or LL; MF, fr. LL *concupiscibilis,* fr. L *concupiscere*] (14c) : LUSTFUL, DESIROUS

con-cur \kən-'kər, kän-\ *vi* **con-curred; con-cur-ring** [ME *concurren,* fr. L *concurrere,* fr. *com-* + *currere* to run — more at CAR] (15c) 1 : to act together to a common end or single effect 2 **a** : APPROVE (~ in a statement) **b** : to express agreement (~ with an opinion) 3 *obs* : to come together : MEET 4 : to happen together : COINCIDE *syn* see AGREE

con-cur-rence \-'kər-ən(t)s, -'kə-rən(t)s\ *n* (15c) 1 **a** : agreement or union in action : COOPERATION **b** (1) : agreement in opinion or design (2) : CONSENT 2 : a coming together : CONJUNCTION 3 : a coincidence of equal powers in law

con-cur-rent \-'kər-ənt, -'kə-rənt\ *adj* [ME, fr. MF & L; MF, fr. L *concurrent-, concurrens,* prp. of *concurrere*] (14c) 1 : operating or occurring at the same time 2 **a** : CONVERGENT; *specif* : meeting or intersecting in a point **b** : running parallel 3 : acting in conjunction 4 : exercised over the same matter or area by two different authorities (~ jurisdiction) — **concurrent** *n* — **con-cur-rent-ly** *adv*

concurrent resolution *n* (1802) : a resolution passed by both houses of a legislative body that lacks the force of law

con-cuss \kən-'kəs\ *vt* [L *concussus,* pp.] (1597) : to affect with or as if with concussion

con-cus-sion \kən-'kəsh-ən\ *n* [MF or L; MF, fr. L *concussion-, concussio,* fr. *concussus,* pp. of *concutere* to shake violently, fr. *com-* + *quatere* to shake] (15c) 1 : AGITATION, SHAKING 2 **a** : a hard blow or collision **b** : a stunning, damaging, or shattering effect from a hard blow; *esp* : a jarring injury of the brain resulting in disturbance of cerebral function — **con-cus-sive** \-'kəs-iv\ *adj*

con-demn \kən-'dem\ *vt* [ME *condemnen,* fr. MF *condemner,* fr. L *condemnare,* fr. *com-* + *damnare* to condemn — more at DAMN] (14c) 1 : to declare to be reprehensible, wrong, or evil usu. after weighing evidence and without reservation 2 **a** : to pronounce guilty : CONVICT **b** : SENTENCE, DOOM 3 : to adjudge unfit for use or consumption 4 : to declare convertible to public use under the right of eminent domain *syn* see CRITICIZE — **con-dem-nable** \-'dem-(n)ə-bəl\ *adj* — **con-dem-na-to-ry** \-nə-,tōr-ē, -,tȯr-\ *adj* — **con-demn-er** \-'dem-ər\ or **con-dem-nor** \kən-'dem-ər, kən-,dem-'nȯ(ə)r, ,kän-\ *n*

con-dem-na-tion \,kän-,dem-'nā-shən, ,dəm-\ *n* (14c) 1 : CENSURE, BLAME 2 : the act of judicially condemning 3 : the state of being condemned 4 : a reason for condemning

con-den-sate \'kän-dən-,sāt, -,den-; kən-'den-\ *n* (ca. 1885) : a product of condensation; *esp* : a liquid obtained by condensation of a gas or vapor (steam ~)

con-den-sa-tion \,kän-,den-'sā-shən, -dən-\ *n* (1603) 1 : the act or process of condensing: as **a** : a chemical reaction involving union between molecules often with elimination of a simple molecule (as water) to form a new more complex compound of often greater molecular weight **b** : a reduction to a denser form (as from steam to water) **c** : compression of a written or spoken work into more concise form 2 : the quality or state of being condensed 3 : a product of condensing; *esp* : an abridgment of a literary work — **con-den-sa-tion-al** \-shnəl, -shən-ᵊl\ *adj*

con-dense \kən-'den(t)s\ *vb* **con-densed; con-dens-ing** [ME *condensen,* fr. MF *condenser,* fr. L *condensare,* fr. *com-* + *densare* to make dense, fr. *densus* dense] *vt* (15c) 1 : to make denser or more compact; *esp* : to subject to condensation ~ *vi* : to undergo condensation *syn* see CONTRACT — **con-dens-able** *also* **con-dens-ible** \-'den(t)-sə-bəl\ *adj*

con-densed *adj* (15c) : reduced to a more compact form; *esp* : having a face that is narrower than that of a typeface not so characterized

condensed milk *n* (1863) : evaporated milk with sugar added

con-dens-er \kən-'den(t)-sər\ *n* (1686) 1 : one that condenses: as **a** : a lens or mirror used to concentrate light on an object **b** : an apparatus in which gas or vapor is condensed 2 : CAPACITOR

con-de-scend \,kän-di-'send\ *vi* [ME *condescenden,* fr. MF *condescendre,* fr. LL *condescendere,* fr. L *com-* + *descendere* to descend] (14c) 1 **a** : to descend to a less formal or dignified level : UNBEND **b** : to waive the privileges of rank 2 : to assume an air of superiority

con-de-scen-dence \-'sen-dən(t)s\ *n* (1638) : CONDESCENSION

con-de-scend-ing *adj* (1707) : showing or characterized by condescension : PATRONIZING — **con-de-scend-ing-ly** \-'sen-diŋ-lē\ *adv*

con-de-scen-sion \,kän-di-'sen-chən\ *n* [LL *condescension-, condescensio,* fr. *condescensus,* pp. of *condescendere*] (1647) 1 : voluntary descent from one's rank or dignity in relations with an inferior 2 : patronizing attitude or behavior

con-dign \kən-'dīn, 'kän-,\ *adj* [ME *condigne,* fr. MF, fr. L *condignus* very worthy, fr. *com-* + *dignus* worthy — more at DECENT] (15c) : DESERVED, APPROPRIATE (~ punishment) — **con-dign-ly** *adv*

con-di-ment \'kän-də-mənt\ *n* [ME, fr. MF, fr. L *condimentum,* fr. *condire* to pickle, fr. *condere* to build, store up, fr. *com-* + *-dere* to put — more at DO] (13c) : something used to enhance the flavor of food; *esp* : a pungent seasoning — **con-di-men-tal** \,kän-də-'ment-ᵊl\ *adj*

con-di-tion \kən-'dish-ən\ *n* [ME *condicion,* fr. MF, fr. L *condicion-, condicio* terms of agreement, condition, fr. *condicere* to agree, fr. *com-* + *dicere* to say, determine — more at DICTION] (14c) 1 **a** : a premise upon which the fulfillment of an agreement depends ; STIPULATION **b** *obs* : COVENANT **c** : a provision making the effect of a legal instrument contingent upon an uncertain event; *also* : the event itself 2 : something essential to the appearance or occurrence of something else : PREREQUISITE: as **a** : an environmental requirement (available oxygen is an essential ~ for animal life) **b** : the subordinate clause of a conditional sentence 3 **a** : a restricting or modifying factor : QUALIFICATION **b** : an unsatisfactory academic grade that may be raised by

RCL011399

Case 5:03-cv-04669-JW Document 238-5 Filed 08/27/2004 Page 3 of 11

however, the earliest and etymologically purest sense of the word. But since the pejorative senses continue to flourish, expressions like "fulsome praise" can be ambiguous; the reader or hearer may not be sure whether sense 1 or sense 3 is intended.

ful·vous \'fůl-vəs, 'fəl-\ *adj* [L *fulvus*; perh. akin to L *flavus* yellow — more at BLUE] (1664) : of a dull brownish yellow : TAWNY

Fu Man·chu mustache \ˌfü-(ˌ)man-'chü-\ *n* [*Fu Manchu*, Chinese villain in stories by "Sax Rohmer" (A. S. Ward †1955)] (ca. 1936) : a long mustache with ends that turn down to the chin

fu·ma·rase \'fyü-mə-ˌrās, -ˌrāz\ *n* (ca. 1936) : an enzyme that catalyzes the interconversion (as in the Krebs cycle) of fumaric acid and malic acid or their salts

fu·ma·rate \-ˌrāt\ *n* (1864) : a salt or ester of fumaric acid

fu·mar·ic acid \fyü-ˌmar-ik-\ *n* [ISV, fr. NL *Fumaria*, genus of herbs, fr. LL, fumitory, fr. L *fumus*] (1876) : a crystalline acid $C_4H_4O_4$ found in various plants or made synthetically and used esp. in making resins

fu·ma·role \'fyü-mə-ˌrōl\ *n* [It *fumarola*, modif. of LL *fumariolum*, fr. L *fumarium* smoke chamber for aging wine, fr. *fumus*] (1811) : a hole in a volcanic region from which hot gases and vapors issue — **fu·ma·rol·ic** \ˌfyü-mə-'rō-lik\ *adj*

¹fum·ble \'fəm-bəl\ *vb* **fum·bled; fum·bling** \-b(ə-)liŋ\ [prob. of Scand origin; akin to Sw *fumla* to fumble] *vi* (1534) 1 **a** : to grope for or handle something clumsily or aimlessly **b** : to make awkward attempts to do or find something ⟨*fumbled* in his pocket for a coin⟩ **c** : to search by trial and error **d** : BLUNDER **2** : to feel one's way or move awkwardly **3 a** : to drop or juggle or fail to play cleanly a grounder **b** : to lose hold of a football while handling or running with it ~ *vt* **1** : to bring about by clumsy manipulation **2 a** : to feel or handle clumsily **b** : to deal with in a blundering way : BUNGLE **3** : to make (one's way) in a clumsy manner **4 a** : MISPLAY ⟨~ a grounder⟩ **b** : to lose hold of (a football) while handling or running — **fum·bler** \-b(ə-)lər\ *n* — **fum·bling·ly** \-b(ə-)liŋ-lē\ *adv*

²fumble *n* (1634) **1** : an act or instance of fumbling **2** : a fumbled ball

¹fume \'fyüm\ *n* [ME, fr. MF *fum*, fr. L *fumus*; akin to OHG *toumen* to be fragrant, Gk *thymos* mind, spirit] (14c) **1 a** : a smoke, vapor, or gas esp. when irritating or offensive ⟨engine exhaust ~s⟩ **b** : an often noxious suspension of particles in a gas (as air) **2** : something (as an emotion) that impairs one's reasoning ⟨sometimes his head gets a little hot with the ~s of patriotism —Matthew Arnold⟩ **3** : a state of excited irritation or anger — usu. used in the phrase *in a fume* — **fumy** \'fyü-mē\ *adj*

²fume *vb* **fumed; fum·ing** *vi* (15c) **1** : to expose to or treat with fumes **2** : to give off in fumes ⟨*fuming* thick black smoke⟩ **3** : to utter while in a state of excited irritation or anger ~ *vi* **1 a** : to emit fumes **b** : to be in a state of excited irritation or anger ⟨he fretted and *fumed* over the delay⟩ **2** : to rise in or as if in fumes

fu·mi·gant \'fyü-mi-gənt\ *n* (1890) : a substance used in fumigating

fu·mi·gate \'fyü-mə-ˌgāt\ *vt* **-gat·ed; -gat·ing** [L *fumigatus*, pp. of *fumigare*, fr. *fumus* + *-igare* (akin to L *agere* to drive) — more at AGENT] (1781) : to apply smoke, vapor, or gas to esp. for the purpose of disinfecting or of destroying pests — **fu·mi·ga·tion** \ˌfyü-mə-'gā-shən\ *n* — **fu·mi·ga·tor** \'fyü-mə-ˌgāt-ər\ *n*

fu·mi·to·ry \'fyü-mə-ˌtōr-ē, -ˌtör-\ *n* [ME *fumeterre*, fr. MF, fr. ML *fumus terrae*, lit., smoke of the earth, fr. L *fumus* + *terrae*, gen. of *terra* earth — more at TERRACE] (14c) : any of a genus (*Fumaria* of the family Fumariaceae, the fumitory family) of erect or climbing herbs; *esp* : a common European herb (*F. officinalis*)

¹fun \'fən\ *n* [E dial. *fun* to hoax, perh. alter. of ME *fonnen*, fr. *fonne* dupe] (1727) **1** : what provides amusement or enjoyment; *specif* : playful often boisterous action or speech ⟨a lively person full of ~⟩ **2** : a mood for finding or making amusement ⟨the teasing was all in ~⟩ **3 a** : AMUSEMENT, ENJOYMENT ⟨sickness takes all the ~ out of life⟩ **b** : derisive jest : SPORT, RIDICULE ⟨made him a figure of ~⟩ **4** : violent or excited activity or argument ⟨let a snake loose in the classroom; then the ~ began⟩

syn FUN, JEST, SPORT, GAME, PLAY mean action or speech that provides amusement or arouses laughter. FUN usu. implies laughter or gaiety but may imply merely a lack of serious or ulterior purpose; JEST implies lack of earnestness in what is said or done and may suggest a hoaxing or teasing; SPORT applies esp. to the arousing of laughter against someone; GAME is close to SPORT, and often stresses mischievous or malicious fun; PLAY stresses the opposition to *earnest* without implying any element of malice or mischief

²fun *vi* **funned; fun·ning** (1833) : to indulge in banter or play : JOKE

³fun *adj* (1846) : providing entertainment, amusement, or enjoyment ⟨a ~ party⟩ ⟨a ~ person to be with⟩

fu·nam·bu·lism \fyü-'nam-byə-ˌliz-əm\ *n* [L *funambulus* ropewalker, fr. *funis* rope + *ambulare* to walk] (1824) **1** : tightrope walking **2** : a show esp. of mental agility — **fu·nam·bu·list** \-list\ *n*

fun and games *n pl but sing or pl in constr* (1920) : light amusement : DIVERSION

¹func·tion \'fəŋ(k)-shən\ *n* [L *function-*, *functio* performance, fr. *functus*, pp. of *fungi* to perform; prob. akin to Skt *bhuñkte* he enjoys] (1533) **1** : professional or official position : OCCUPATION **2** : the action for which a person or thing is specially fitted or used or for which a thing exists : PURPOSE **3** : any of a group of related actions contributing to a larger action; *esp* : the normal and specific contribution of a bodily part to the economy of a living organism **4** : an impressive, elaborate, or formal ceremony or social gathering **5 a** : a mathematical correspondence that assigns exactly one element of one set to each element of the same or another set **b** : a variable (as a quality, trait, or measurement) that depends on and varies with another ⟨height is a ~ of age⟩ **6** : characteristic behavior of a chemical compound due to a particular reactive unit — **func·tion·less** \-ləs\ *adj*

syn FUNCTION, OFFICE, DUTY, PROVINCE mean the acts or operations expected of a person or thing. FUNCTION implies a definite end or purpose that the one in question serves or a particular kind of work it is intended to perform ⟨the *function* of language is two-fold: to communicate emotion and to give information —Aldous Huxley⟩ OFFICE is typically applied to the function or service expected of a person by reason of his trade or profession or his special relationship to others ⟨they exercise the *offices* of the judge; the priest, the counsellor — W.E. Gladstone⟩ DUTY applies to a task or responsibility imposed by one's occupation, rank, status, or calling ⟨it is the judicial *duty* of the court, to examine the whole case —R. B. Taney⟩ PROVINCE applies to a function, office, or duty that naturally or logically falls to one ⟨nursing does not belong to a man; it is not his *province* —Jane Austen⟩

²function *vi* **func·tioned; func·tion·ing** \-sh(ə-)niŋ\ (1856) **1** : to have a function : SERVE ⟨an attributive noun ~s as an adjective⟩ **2** : to be in action : OPERATE ⟨a government ~s through numerous divisions⟩

func·tion·al \'fəŋ(k)-shnəl, -shən-²l\ *adj* (1631) **1 a** : of, connected with, or being a function **b** : affecting physiological or psychological functions but not organic structure ⟨~ heart disease⟩ **2** : used to contribute to the development or maintenance of a larger whole ⟨~ and practical school courses⟩; *also* : designed or developed chiefly from the point of view of use ⟨~ clothing⟩ **3** : performing or able to perform a regular function — **func·tion·al·ity** \ˌfəŋ(k)-shə-'nal-ət-ē\ *n* — **func·tion·al·ly** \'fəŋ(k)-shnə-lē, -shən-²l-ē\ *adv*

functional calculus *n* (1933) : PREDICATE CALCULUS

functional group *n* (ca. 1939) : a characteristic reactive unit of a chemical compound esp. in organic chemistry

functional illiterate *n* (1947) : a person having had some schooling but not meeting a minimum standard of literacy

func·tion·al·ism \'fəŋ(k)-shnə-ˌliz-əm, -shən-²l-ˌiz-\ *n* (1914) **1** : a philosophy of design (as in architecture) holding that form should be adapted to use, material, and structure **2** : a theory that stresses the interdependence of the patterns and institutions of a society and their interaction in maintaining cultural and social unity **3** : a doctrine or practice that emphasizes practical utility or functional relations — **func·tion·al·ist** \-shnə-ləst, -shən-²l-əst\ *n* — **functionalist** *or* **func·tional·is·tic** \ˌfən(k)-shnə-'lis-tik, -shən-²l-'is-\ *adj*

functional shift *n* (1942) : the process by which a word or form comes to be used in a second or third grammatical function ⟨the *functional shift* of "go" from verb to adjective as in "all systems are go"⟩

func·tion·ary \'fəŋ(k)-shə-ˌner-ē\ *n, pl* **-ar·ies** (1791) **1** : one who serves in a certain function **2** : one holding office in a government or political party

function word *n* (1940) : a word (as a preposition, auxiliary verb, or conjunction) expressing primarily grammatical relationship

func·tor \'fəŋ(k)-tər\ *n* (1935) : something that performs a function or an operation

¹fund \'fənd\ *n* [L *fundus* bottom, piece of landed property — more at BOTTOM] (1682) **1** : an available quantity of material or intangible resources : SUPPLY **2 a** : a sum of money or other resources whose principal or interest is set apart for a specific objective **b** : money on deposit on which checks or drafts can be drawn — usu. used in pl. **c** : CAPITAL **d** *pl* : the stock of the British national debt — usu. used with *the* **3** *pl* : available pecuniary resources **4** : an organization administering a special fund

²fund *vt* (1776) **1 a** : to make provision of resources for discharging the interest or principal of **b** : to provide funds for ⟨a science program federally ~*ed*⟩ **2** : to place in a fund : ACCUMULATE **3** : to convert into a debt that is payable either at a distant date or at no definite date and that bears a fixed interest ⟨~ a floating debt⟩

fun·da·ment \'fən-də-mənt\ *n* [ME, fr. OF *fondement*, fr. L *fundamentum*, fr. *fundare* to found, fr. *fundus*] (13c) **1** : an underlying ground, theory, or principle **2 a** : BUTTOCKS **b** : ANUS **3** : the part of a load surface that has not been altered by human activities

¹fun·da·men·tal \ˌfən-də-'ment-²l\ *adj* (15c) **1 a** : serving as an original or generating source : PRIMARY ⟨a discovery ~ to scientific progress⟩ **b** : serving as a basis supporting existence or determining essential structure or function : BASIC **2 a** : of or relating to essential structure, function, or facts : RADICAL ⟨~ change⟩; *specif* : of or dealing with general principles rather than practical application ⟨~ science⟩ **b** : adhering to fundamentalism **3** : of, relating to, or produced by the lowest component of a complex vibration **4** : of central importance : PRINCIPAL ⟨~ purpose⟩ **5** : belonging to one's innate or ingrained characteristics : DEEP-ROOTED ⟨hard to spoil his ~ good humor⟩ *syn* see ESSENTIAL — **fun·da·men·tal·ly** \-²l-ē\ *adv*

²fundamental *n* (1633) **1** : something fundamental; *esp* : one of the minimum constituents without which a thing or a system would not be what it is **2 a** : the principal musical tone produced by vibration (as of a string or column of air) on which a series of higher harmonics is based **b** : the root of a chord **3** : the harmonic component of a complex wave that has the lowest frequency and commonly the greatest amplitude

fun·da·men·tal·ism \-²l-ˌiz-əm\ *n* (1922) **1 a** *often cap* : a movement in 20th century Protestantism emphasizing the literally interpreted Bible as fundamental to Christian life and teaching **b** : the beliefs of this movement **c** : adherence to such beliefs **2** : a movement or attitude stressing strict and literal adherence to a set of basic principles — **fundamentalist** *or* **fun·da·men·tal·is·tic** \-ˌent-²l-'is-tik\ *adj*

fundamental law *n* (ca. 1914) : the organic or basic law of a political unit as distinguished from legislative acts; *specif* : CONSTITUTION

fundamental particle *n* (ca. 1934) : ELEMENTARY PARTICLE

fundamental tissue *n* (1887) : plant tissue other than dermal and vascular tissues that consists typically of relatively undifferentiated parenchymatous and supportive cells

fun·dic \'fən-dik\ *adj* (1927) : of or relating to a fundus

fund–rais·er \'fən-ˌdrā-zər\ *n* (1957) **1** : a person employed to raise funds **2** : a social event (as a cocktail party) held for the purpose of raising funds

fund–rais·ing \-ziŋ\ *n* (1940) : the organized activity of raising funds (as for an institution or political cause)

fun·dus \'fən-dəs\ *n, pl* **fun·di** \-ˌdī, -ˌdē\ [NL, fr. L, bottom] (1754) : the bottom of or part opposite the aperture of the internal surface of a hollow organ: as **a** : the greater curvature of the stomach **b** : the lower back part of the bladder **c** : the large upper end of the uterus **d** : the part of the eye opposite the pupil

¹fu·ner·al \'fyün-(ə-)rəl\ *adj* [ME, fr. LL *funeralis*, fr. L *funer-*, *funus* funeral (n.); perh. akin to ON *deyja* to die — more at DIE] (14c) **1** : of, relating to, or constituting a funeral **2** : FUNEREAL 2

²funeral *n* [ME *funerelles* (pl.), fr. MF *funerailles* (pl.), fr. ML *funeralia* (pl.), fr. LL, neut. pl. of *funeralis*, adj.] (1512) **1** : the obsequies held for a dead person usu. before burial or cremation **2** *chiefly dial* : a

RCL011400

Case 5:03-cv-04669-JW Document 238-5 Filed 08/27/2004 Page 4 of 11

…dē-ˌkant\ *n* [ME *dyscant*, fr. ONF & ML; ONF *descant*, fr. … fr. L *dis-* + *cantus* song — more at CHANT] (14c) **1** *a* … counterpoint sung above the plainsong of the tenor *b* … composing or improvising contrapuntal part music; *also* … so composed or improvised *c* : SOPRANO, TREBLE *d* : … counterpoint to a simple melody sung typically by some … sopranos **2** : discourse or comment on a theme

…kant, des-ˌ, dis-\ *vi* (15c) **1** : to sing or play a descant; … **2** : COMMENT, DISCOURSE

…-ˈsend\ *vb* [ME *descendan*, fr. OF *descendre*, fr. L *descendere* : *scandere* to climb — more at SCAN] *vi* (14c) **1** : to pass … place or level to a lower one ⟨~ed from the platform⟩ **2** … discussion from what is logically prior or more comprehen… **3** : to come down from a stock or source : DERIVE — usu. … (was ~ed from an ancient family) **b** : to pass by in… (an heirloom that has ~ed in the family) **c** : to pass by … (songs ~ed from early ballads) **4** : to incline, lead, or … downward (the road ~s to the river) **5** : to swoop or pounce … make a sudden attack ⟨the plague ~ed upon them⟩ **6** : to … in a sequence or gradation from higher to lower or from more … nearer or more recent **7** *a* : to lower oneself in status or … DEGRADE **b** : to worsen and sink in condition or estimation ~ *vt* … move, or climb down or down along **2** : to extend down

…ˈsen-də-bəl\ *adj*

…dē-ˈsen-dənt\ *adj* [MF & L; MF *descendant*, fr. L *descendent-*, *descendens*, prp. of *descendere*] (1572) **1** … or directed downward **2** : proceeding from an ancestor or …

…ˈsen-dənt\ *n* [F & L; F *descendant*, fr. LL *descendent-*, … fr. L] (1600) **1** : one descended from another or from a … stock **2** : one deriving directly from a precursor or proto…

…ˈsen-ˌdōr, -ˌdȯr\ *n* (1802) : the part of a lowercase letter … that descends below the main body of the letter; *also* : a letter …

…ˈsen-chən\ *n, archaic* (14c) : DESCENT 2

…dē-ˈsent\ *n* [ME, fr. MF *descente*, fr. OF *descendre*] (14c) **1** *a* … derivation from an ancestor : BIRTH, LINEAGE ⟨of French ~⟩ **b** … succession or devolution of an estate by inheritance usu. in the … descending line **c** : the fact or process of originating from an ancestral … stock **2** : the shaping or development in nature and character by … modification from a source : DERIVATION **2** : the act or process of … descending **3** : a step downward in a scale of gradation; *specif* : one … gradation in an ancestral line or genealogical scale **4** *a* : an inclina… downward : SLOPE **b** : a descending way (as a downgrade or stair… *obs* : the lowest part **5** *a* : ATTACK, INVASION **b** : a sudden … discomforting appearance (as for a visit) **6** : a downward step (as in … station or value) : DECLINE ⟨~ of the family to actual poverty⟩

de-scribe \dē-ˈskrīb\ *vt* **de-scribed; de-scrib-ing** [L *describere*, fr. *de-* + *scribere* to write — more at SCRIBE] (15c) **1** : to represent or give an account of in words ⟨~ a picture⟩ **2** : to represent by a figure, model, or picture : DELINEATE **3** *obs* : DISTRIBUTE **4** : to trace or traverse the outline of ⟨~ a circle⟩ **5** *archaic* : OBSERVE, PERCEIVE — **de-scrib-able** \-ˈskrī-bə-bəl\ *adj* — **de-scrib-er** *n*

de-scrip-tion \dē-ˈskrip-shən\ *n* [ME *descripcioun*, fr. MF & L; MF *description*, fr. L *description-*, *descriptio*, fr. *descriptus*, pp. of *describere*] (14c) **1** *a* : an act of describing; *specif* : discourse intended to give a mental image of something experienced (as a scene, person, or sensation) **b** : a descriptive statement or account ⟨a fascinating ~ of his adventures⟩ **2** : kind or character esp. as determined by salient features ⟨opposed to any tax of so radical a ~⟩ *syn* see TYPE

de-scrip-tive \dē-ˈskrip-tiv\ *adj* (1751) **1** : serving to describe ⟨a ~ account⟩ **2** : referring to, constituting, or grounded in matters of observation or experience ⟨the ~ basis of science⟩ **3** *of a modifier* : expressing the quality, kind, or condition of what is denoted by the modified term ⟨*hot* in "hot water" is a ~ adjective⟩ **b** : NONRESTRICTIVE **4** : of, relating to, or dealing with the structure of a language at a particular time usu. with exclusion of historical and comparative data ⟨~ linguistics⟩ — **de-scrip-tive-ly** *adv* — **de-scrip-tive-ness** *n*

de-scriptor \dē-ˈskrip-tər\ *n* (1951) : something (as an index term) used to identify an item (as a subject of document) esp. in an information retrieval system

de-scry \dē-ˈskrī\ *vt* **de-scried; de-scry-ing** [ME *descrien*, fr. MF *descrier* to proclaim, decry] (14c) **1** *a* : to catch sight of **b** : to find out : DISCOVER **2** *obs* : to make known : REVEAL

descry *n, obs* (1605) : discovery or view from afar

Des-de-mo-na \ˌdez-də-ˈmō-nə\ *n* : the wife of Othello in Shakespeare's *Othello*

des-e-crate \ˈdes-i-ˌkrāt\ *vt* **-crat-ed; -crat-ing** [*de-* + *-secrate* (as in *consecrate*)] (1612) **1** : to violate the sanctity of : PROFANE **2** : to treat irreverently or contemptuously often in a way that provokes outrage ⟨the part of others ⟨the kind of shore development . . . that has *desecrated* so many waterfronts — John Fischer⟩⟩ — **des-e-crat-er** *or* **des-e-cra-tor** \-ˌkrāt-ər\ *n*

des-e-cration \ˌdes-i-ˈkrā-shən\ *n* (1717) : an act or instance of desecrating : the state of being desecrated

de-segre-gate \(ˌ)dē-ˈseg-ri-ˌgāt\ *vt* (1952) : to eliminate segregation in; *specif* : to free of any law, provision, or practice requiring isolation of the members of a particular race in separate units ~ *vi* : to bring about desegregation

de-segre-ga-tion \(ˌ)dē-ˌseg-ri-ˈgā-shən\ *n* (1951) **1** : the act or process or an instance of desegregating **2** : the state of being desegregated

de-se-lect \ˌdē-sə-ˈlekt\ *vt* (1965) : to dismiss (a trainee) from a training program

de-sen-si-tize \(ˌ)dē-ˈsen(t)-sə-ˌtīz\ *vt* (1898) **1** : to make (a sensitized or hypersensitive individual) insensitive or nonreactive to a sensitizing agent **2** : to make (a photographic material) less sensitive or completely insensitive to radiation **3** : to make emotionally insensitive or callous; *specif* : to extinguish an emotional response (as of fear, anxiety, or guilt) to stimuli that formerly induced it — **de-sen-si-ti-za-tion** \(ˌ)dē-ˌsen-sət-ə-ˈzā-shən, -ˌsen-stə-ˈzā-\ *n* — **de-sen-si-tiz-er** \(ˌ)dē-ˈsen-sə-ˌtī-zər\ *n*

desert \ˈdez-ərt\ *n* [ME, fr. OF, fr. LL *desertum*, fr. L, neut. of *desertus*, pp. of *deserere* to desert, fr. *de-* + *serere* to join together — more at SERIES] (13c) **1** *a* : arid barren land; *esp* : a tract incapable of supporting any considerable population without an artificial water supply **b** : an area of water apparently devoid of life **2** *archaic* : a wild uninhabited and uncultivated tract **3** : a desolate or forbidding area ⟨lost in a ~ of doubt⟩ — **desert-like** \-ˌlīk\ *adj*

²desert \ˈdez-ərt\ *adj* (13c) **1** : desolate and sparsely occupied or unoccupied ⟨a ~ island⟩ **2** : of or relating to a desert **3** *archaic* : FORSAKEN

³de-sert \dē-ˈzərt\ *n* [ME *deserte*, fr. OF, fr. fem. of *desert*, pp. of *deservir* to deserve] (13c) **1** : the quality or fact of deserving reward or punishment **2** : deserved reward or punishment — usu. used in plural ⟨got his just ~s⟩ **3** : EXCELLENCE, WORTH

⁴de-sert \dē-ˈzərt\ *vb* [F *déserter*, fr. LL *desertare*, fr. *desertus*] *vt* (1603) **1** : to withdraw from or leave usu. without intent to return **2** *a* : to leave in the lurch ⟨~ a friend in trouble⟩ **b** : to abandon (military service) without leave ~ *vi* : to quit one's post, allegiance, or service without leave or justification; *esp* : to absent oneself from military duty without leave and without intent to return *syn* see ABANDON — **de-sert-er** *n*

de-sert-i-fi-ca-tion \dē-ˌzərt-ə-fə-ˈkā-shən\ *n* [*desert* + *-ification* (as in *saponification*)] (1974) : the process of becoming arid land or desert (as from land mismanagement or climate change)

de-ser-tion \dē-ˈzər-shən\ *n* (1591) **1** : an act of deserting; *esp* : the abandonment without consent or legal justification of a person, post, or relationship and the associated duties and obligations ⟨sued for divorce on grounds of ~⟩ **2** : a state of being deserted or forsaken

desert locust *n* (1944) : a destructive migratory locust (*Schistocerca gregaria*) of southwestern Asia and parts of northern Africa

desert soil *n* (ca. 1938) : a soil that develops under sparse shrub vegetation in warm to cool arid climates with a light-colored surface soil usu. underlain by calcareous material and a hardpan layer

de-serve \dē-ˈzərv\ *vb* **de-served; de-serv-ing** [ME *deserven*, fr. OF *deservir*, fr. L *deservire* to serve zealously, fr. *de-* + *servire* to serve] *vt* (13c) : to be worthy of : MERIT ⟨~s another chance⟩ ~ *vi* : to be worthy, fit, or suitable for some reward or requital ⟨have become recognized as they ~ —T. S. Eliot⟩ — **de-serv-er** *n*

de-served \-ˈzərvd\ *adj* (ca. 1552) : of, relating to, or being that which one deserves ⟨a ~ reputation⟩ — **de-served-ly** \-ˈzər-vəd-lē, -ˈzərv-dlē\ *adv* — **de-served-ness** \-ˈzər-vəd-nəs, -ˈzərv(d)-nəs\ *n*

¹de-serv-ing \-ˈzər-viŋ\ *n* (14c) : DESERT, MERIT ⟨reward the proud according to their ~s —Charles Kingsley⟩

²deserving *adj* (1576) : MERITORIOUS, WORTHY; *specif* : meriting financial aid ⟨scholarships for ~ students⟩

de-sex \(ˌ)dē-ˈseks\ *vt* (1911) : CASTRATE, SPAY

de-sex-u-al-ize \(ˌ)dē-ˈsekshˌ(ə-)wə-ˌlīz, -ˈsek-shə-ˌlīz\ *vt* (1894) **1** : to deprive of sexual characters or power **2** : to divest of sexual quality — **de-sex-u-al-i-za-tion** \(ˌ)dē-ˌsekshˌ(ə-)wə-lə-ˈzā-shən, -ˈsek-shə-lə-\ *n*

des-ha-bille \ˌdes-ə-ˈbē(ə)l, -ˈbīl, -ˈbē\ *var of* DISHABILLE

des-ic-cant \ˈdes-i-kənt\ *n* (1676) : a drying agent (as calcium chloride)

des-ic-cate \ˈdes-i-ˌkāt\ *vb* **-cat-ed; -cat-ing** [L *desiccatus*, pp. of *desiccare* to dry up, fr. *de-* + *siccare* to dry, fr. *siccus* dry — more at SACK] *vt* (1575) **1** : to dry up **2** : to preserve (a food) by drying : DEHYDRATE **3** : to drain of emotional or intellectual vitality ~ *vi* : to become dried up — **des-ic-ca-tion** \ˌdes-i-ˈkā-shən\ *n* — **de-sic-ca-tive** \ˈdes-i-ˌkāt-iv, dī-ˈsik-ət-\ *adj* — **des-ic-ca-tor** \ˈdes-i-ˌkāt-ər\ *n*

de-sid-er-ate \dī-ˈsid-ə-ˌrāt, -ˈzid-\ *vt* **-at-ed; -at-ing** [L *desideratus*, pp. of *desiderare* to desire] (1645) : to entertain or express a wish to have or attain — **de-sid-er-a-tion** \-ˌsid-ə-ˈrā-shən, -ˌzid-\ *n* — **de-sid-er-a-tive** \-ˈsid-ə-ˌrāt-iv, -ˈsid-(ə-)rət-, -ˈzid-\ *adj*

de-sid-er-a-tum \-ˌsid-ə-ˈrāt-əm, -ˌzid-, -ˈrät-\ *n, pl* **-ta** \-ə\ [L, neut. of *desideratus*] (1652) : something desired as essential

¹de-sign \dī-ˈzīn\ *vb* [MF *designer*, fr. L *designare*, fr. *de-* + *signare* to mark, mark out — more at SIGN] *vt* (1548) **1** *a* : to conceive and plan out in the mind ⟨he ~ed the perfect crime⟩ **b** : to have as a purpose : INTEND ⟨he ~ed to excel in his studies⟩ **c** : to devise for a specific function or end ⟨a book ~ed primarily as a college textbook⟩ **2** *archaic* : to indicate with a distinctive mark, sign, or name **3** *a* : to make a drawing, pattern, or sketch of **b** : to draw the plans for **c** : to create, fashion, execute, or construct according to plan : DEVISE, CONTRIVE ~ *vi* **1** : to conceive or execute a plan **2** : to draw, lay out, or prepare a design — **de-sign-ed-ly** \-ˈzī-nəd-lē\ *adv*

²design *n* (1588) **1** *a* : a particular purpose held in view by an individual or group ⟨he has ambitious ~s for his son⟩ **b** : deliberate purposive planning ⟨battle was joined . . . more by accident than ~ —John Buchan⟩ **2** : a mental project or scheme in which means to an end are laid down **3** *a* : a deliberate undercover project or scheme; *pl* : aggressive or evil intent — used with *on* or *against* ⟨he has ~s on the money⟩ **4** : a preliminary sketch or outline showing the main features of something to be executed : DELINEATION **5** *a* : an underlying scheme that governs functioning, developing, or unfolding : PATTERN, MOTIF ⟨the general ~ of the epic⟩ **b** : a plan or protocol for carrying out or accomplishing something (esp. a scientific experiment); *also* : the process of preparing this **6** : the arrangement of elements or details in a product or work of art **7** : a decorative pattern **8** : the creative art of executing aesthetic or functional designs *syn* see INTENTION, PLAN

¹des-ig-nate \ˈdez-ig-ˌnāt, -nət\ *adj* [L *designatus*, pp. of *designare*] (1646) : chosen for an office but not yet installed ⟨ambassador ~⟩

²des-ig-nate \-ˌnāt\ *vt* **-nat-ed; -nat-ing** (1791) **1** : to indicate and set apart for a specific purpose, office, or duty **2** *a* : to point out the location of ⟨a marker *designating* the crest of the floodwaters⟩ **b** : INDICATE ⟨any task *designated* by the employer⟩ **c** : to distinguish as to class ⟨the area we ~ as that of spiritual values —J. B. Conant⟩ **d** : SPECIFY, STIPULATE **3** : DENOTE **4** : to call by a distinctive title, term,

RCL011401

Case 5:03-cv-04669-JW Document 238-5 Filed 08/27/2004 Page 5 of 11

eminence or preference; SPECIFIC implies a quality or character distinguishing a kind or a species; PARTICULAR stresses the distinctness of something as an individual; INDIVIDUAL implies unequivocal reference to one of a class or group.

²special *n* (ca.1909) 1 : something (as a television program) that is not part of a regular series 2 : one that is used for a special service or occasion ⟨caught the commuter ~ to work⟩

special assessment *n* (1875) : a specific tax levied on private property to meet the cost of public improvements that enhance the value of the property

special delivery *n* (1886) : expedited messenger delivery of mail matter for an extra fee

special district *n* (1950) : a political subdivision of a state established to provide a single public service (as water supply or sanitation) within a specific geographical area

special drawing rights *n* (1967) : a means of exchange used by governments to settle their international indebtedness

special effects *n pl* (1944) : visual or sound effects introduced into a motion picture or a taped television production during laboratory processing

Special Forces *n pl* (1962) : a branch of the army composed of men specially trained in guerrilla warfare

special handling *n* (1928) : the handling of parcel-post or fourth-class mail as first-class but not as special-delivery matter for an extra postal fee

special interest *n* (1910) : a person or group seeking to influence legislative or government policy to further often narrowly defined interests; *esp* : LOBBY

spe·cial·ism \'spesh-ə-ˌli-zəm\ *n* (1856) 1 : specialization in an occupation or branch of learning 2 : a field of specialization : SPECIALTY

spe·cial·ist \'spesh-(ə-)list\ *n* (1856) 1 : one who devotes himself to a special occupation or branch of learning 2 : any of four enlisted ranks in the army corresponding to the grades of corporal through sergeant first class — **specialist** *or* **special-is·tic** \ˌspesh-ə-'lis-tik\ *adj*

spe·ci·al·i·ty \ˌspesh-ē-'al-ət-ē\ *n, pl* **-ties** (15c) 1 : a special mark or quality 2 : a special object or class of objects 3 **a** : a special aptitude or skill **b** : a particular occupation or branch of learning

spe·cial·iza·tion \ˌspesh-(ə-)lə-'zā-shən\ *n* (1843) 1 : a making or becoming specialized 2 **a** : structural adaptation of a body part to a particular function or of an organism for life in a particular environment **b** : a body part or an organism adapted by specialization

spe·cial·ize \'spesh-ə-ˌlīz\ *vb* **-lized; -liz·ing** *vi* (1613) 1 : to make particular mention of : PARTICULARIZE 2 : to apply or direct to a specific end or use ⟨*specialized* his study⟩ ~ *vi* 1 : to concentrate one's efforts in a special activity or field 2 : to undergo specialization; *esp* : to change adaptively ⟨the sloth became highly *specialized* in the course of evolution⟩

specialized *adj* (1853) 1 : designed or fitted for one particular purpose or occupation ⟨~ personnel⟩ 2 : characterized by or exhibiting biological specialization; *esp* : highly differentiated esp. in a particular direction or for a particular end

special jury *n* (1730) : a jury chosen by the court on request from a list of better educated or presumably more intelligent prospective jurors for a case involving complicated issues of fact or serious felonies — called also *blue-ribbon jury*

special pleading *n* (1684) 1 : the allegation of special or new matter to offset the effect of matter pleaded by the opposite side and admitted, as distinguished from a direct denial of the matter pleaded 2 : misleading argument that presents one point or phase as if it covered the entire question at issue

special theory of relativity (1924) : RELATIVITY 3a

spe·cial·ty \'spesh-əl-tē\ *n, pl* **-ties** [ME *specialte*, fr. MF *especialté*, fr. LL *specialitat-, specialitas*, fr. L *specialis* special] (14c) 1 : a distinctive mark or quality 2 **a** : a special object or class of objects: as (1) : a legal agreement embodied in a sealed instrument (2) : a product of a special kind or of special excellence ⟨fried chicken was father's ~⟩ **b** : the state of being special, distinctive, or peculiar 3 : something in which one specializes

spe·ci·a·tion \ˌspē-s(h)ē-'ā-shən\ *n* (ca. 1900) : the process of biological species formation — **spe·ci·ate** \'spē-s(h)ē-ˌāt\ *vi* — **spe·ci·a·tion·al** \ˌspē-s(h)ē-'ā-shnəl, -shən-'l\ *adj*

¹spe·cie \'spē-shē; -sē\ *n* [fr. *in specie*, fr. L, in kind] (1617) : money in coin — **in specie** : in the same or like form or kind ⟨ready to return insult *in specie*⟩; *also* : in coin

²specie *n* [back-formation fr. *species* (taken as a pl.)] *substand* (1711) : SPECIES

¹spe·cies \'spē-(ˌ)shēz, -(ˌ)sēz\ *n, pl* **species** [L, appearance, kind, species — more at SPY] (1551) 1 **a** : a class of individuals having common attributes and designated by a common name; *specif* : a logical division of a genus or more comprehensive class **b** : KIND, SORT **c** : the human race : human beings — often used with *the* ⟨survival of the ~ in the nuclear age⟩ **d** (1) : a category of biological classification ranking immediately below the genus or subgenus, comprising related organisms or populations potentially capable of interbreeding, and being designated by a binomial that consists of the name of a genus followed by a Latin or latinized uncapitalized noun or adjective agreeing grammatically with the genus name (2) : an individual or kind belonging to a biological species **e** : a particular kind of atomic nucleus, atom, molecule, or ion 2 : the consecrated eucharistic elements of the Roman Catholic or Eastern Orthodox Eucharist 3 **a** : a mental image; *also* : a sensible object **b** : an object of thought correlative with a natural object

²species *adj* (1899) : belonging to a biological species as distinguished from a horticultural variety ⟨a ~ rose⟩

spe·cies·ism \'spē-shē-ˌsiz-əm, -sē-\ *n* [*species* + *-ism* (as in *racism*)] (1973) : prejudice or discrimination based on species; *esp* : discrimination against animals

¹spe·cif·ic \spi-'sif-ik\ *adj* [LL *specificus*, fr. L *species*] (1631) 1 **a** : constituting or falling into a specifiable category **b** : sharing or being those properties of something that allow it to be referred to a particular category 2 **a** : restricted to a particular individual, situation, relation, or effect ⟨a disease ~ to horses⟩ **b** : exerting a distinctive influence (as on a body part or a disease) ⟨~ antibodies⟩ 3 : free from ambiguity : ACCURATE ⟨a ~ statement of faith⟩ 4 : of, relating to, or constituting a species and esp. a biologic species 5 **a** : being any of various arbitrary physical constants and esp. one relating a quantitative attribute to unit mass, volume, or area **b** : imposed at a fixed rate per unit (as of weight or count) ⟨~ import duties⟩ — compare AD VALOREM *syn* see SPECIAL, EXPLICIT — **spe·cif·i·cal·ly** \-i-k(ə-)lē\ *adv*

²specific *n* (1661) 1 **a** : something peculiarly adapted to a purpose or use **b** : a drug or remedy having a specific mitigating effect on a disease 2 **a** : a characteristic quality or trait **b** : DETAIL, PARTICULARS — usu. used in pl. ⟨haggling over the legal and financial ~s of independence —*Time*⟩ **c** *pl* : SPECIFICATION 2a

spec·i·fi·ca·tion \ˌspes-(ə-)fə-'kā-shən\ *n* (1615) 1 : the act or process of specifying 2 **a** : a detailed precise presentation of something or of a plan or proposal for something — usu. used in pl. **b** : a statement of legal particulars (as of charges or of contract terms); *also* : a single item of such statement **c** : a written description of an invention for which a patent is sought

specific epithet *n* (1947) : the Latin or latinized noun or adjective that follows the genus name in a taxonomic binomial

specific gravity *n* (1666) : the ratio of the density of a substance to the density of some substance (as pure water or hydrogen) taken as a standard when both densities are obtained by weighing in air

specific heat *n* (1832) 1 : the ratio of the quantity of heat required to raise the temperature of a body one degree to that required to raise the temperature of an equal mass of water one degree 2 : the heat in calories required to raise the temperature of one gram of a substance one degree centigrade

specific impulse *n* (1947) : the thrust produced per unit rate of consumption of the propellant that is usu. expressed in pounds of thrust per pound of propellant used per seoond and that is a measure of the efficiency of a rocket engine

spec·i·fic·i·ty \ˌspes-ə-'fis-ət-ē\ *n* (1876) : the quality or condition of being specific: as **a** : the condition of being peculiar to a particular individual or group of organisms ⟨host ~ of a parasite⟩ **b** : the condition of participating in or catalyzing only one or a few chemical reactions ⟨the ~ of an enzyme⟩

specific performance *n* (1873) 1 : the performance of a legal contract strictly or substantially according to its terms 2 : an equitable remedy enjoining specific performance

spec·i·fy \'spes-ə-ˌfī\ *vt* **-fied; -fy·ing** [ME *specifien*, fr. MF *specifier*, fr. LL *specificare*, fr. *specificus*] (14c) 1 : to name or state explicitly or in detail 2 : to include as an item in a specification — **spec·i·fi·able** \-fī-ə-bəl\ *adj* — **spec·i·fi·er** \-ˌfī(-ə)r\ *n*

spec·i·men \'spes-(ə-)mən\ *n* [L, fr. *specere* to look at, look — more at SPY] (1610) 1 : an item or part typical of a group or whole 2 **a** : something that obviously belongs to a particular category but is noticed by reason of an individual distinguishing characteristic **b** : PERSON, INDIVIDUAL ⟨he's a tough ~⟩ *syn* see INSTANCE

spe·ci·os·i·ty \ˌspē-shē-'äs-ət-ē\ *n* (1608) : the quality or state of being specious : SPECIOUSNESS

spe·cious \'spē-shəs\ *adj* [ME, fr. L *speciosus* beautiful, plausible, fr. *species*] (15c) 1 *obs* : SHOWY 2 : having deceptive attraction or allure 3 : having a false look of truth or genuineness : SOPHISTIC — **spe·cious·ly** *adv* — **spe·cious·ness** *n*

¹speck \'spek\ *n* [ME *specke*, fr. OE *specca*] (bef. 12c) 1 : a small discoloration or spot esp. from stain or decay 2 : a very small amount : BIT 3 : something marked or marred with specks — **specked** \'spekt\ *adj*

²speck *vt* (1580) : to produce specks on or in

¹speck·le \'spek-əl\ *n* [ME; akin to OE *specca*] (15c) : a little speck (as of color)

²speckle *vt* **speck·led; speck·ling** \-(ə-)liŋ\ (ca. 1570) 1 : to mark with speckles 2 : to be distributed in or on like speckles

speckled perch *n* (1888) : BLACK CRAPPIE

speckled trout *n* (1805) 1 : BROOK TROUT 2 : SPOTTED SEA TROUT

¹specs \'speks\ *n pl* [contr. of *spectacles*] (1807) : EYEGLASSES

²specs *n pl* [by contr.] (1942) : SPECIFICATIONS

spec·ta·cle \'spek-ti-kəl *also* -ˌtik-əl\ *n* [ME, fr. MF, fr. L *spectaculum*, fr. *spectare* to watch, fr. *spectus*, pp. of *specere* to look, look at — more at SPY] (14c) 1 **a** : something exhibited to view as unusual, notable, or entertaining; *esp* : an eye-catching or dramatic public display **b** : an object of curiosity or contempt ⟨made a ~ of herself⟩ 2 *pl* : GLASSES 3 : something (as natural markings on an animal) suggesting a pair of glasses

spec·ta·cled \-ti-kəld, -ˌtik-əld\ *adj* (1607) 1 : having or wearing spectacles 2 : having markings suggesting a pair of spectacles ⟨a ~ alligator⟩

¹spec·tac·u·lar \spek-'tak-yə-lər, spək-\ *adj* [L *spectaculum*] (1682) : of, relating to, or constituting a spectacle : STRIKING, SENSATIONAL ⟨a ~ display of fireworks⟩ — **spec·tac·u·lar·ly** *adv*

²spectacular *n* (1890) : something that is spectacular

spec·tate \'spek-ˌtāt\ *vi* **spec·tat·ed; spec·tat·ing** [back-formation fr. *spectator*] (1709) : to be present as a spectator (as at a sports event)

spec·ta·tor \'spek-ˌtāt-ər, spek-'\ *n* [L, fr. *spectatus*, pp. of *spectare* to watch] (1586) : one who looks on or watches — **spectator** *adj*

spec·ter *or* **spec·tre** \'spek-tər\ *n* [F *spectre*, fr. L *spectrum* appearance, specter, fr. *specere* to look, look at — more at SPY] (1605) 1 : a visible disembodied spirit : GHOST 2 : something that haunts or perturbs the mind : PHANTASM ⟨the ~ of hunger⟩

spec·ti·no·my·cin \ˌspek-tə-nō-'mīs-ᵊn\ *n* [NL, fr. *spectabilis* + *-in* + *-o-* + *-mycin*] (ca. 1964) : a white crystalline broad-spectrum antibiotic $C_{14}H_{24}N_2O_7$ produced by a bacterium (*Streptomyces spectabilis*) that is used clinically esp. in the form of its hydrochloride to treat gonorrhea

spec·tral \'spek-trəl\ *adj* (1815) 1 : of, relating to, or suggesting a specter : GHOSTLY 2 : of, relating to, or made by a spectrum — **spec·tral·ly** \'spek-trə-lē\ *adv*

spectral line *n* (1902) : one of a series of linear images of the narrow slit of a spectrograph or similar instrument corresponding to a component of the spectrum of the radiation emitted by a particular source

spectro- *comb form* [NL *spectrum*] : spectrum ⟨*spectroscope*⟩

spec·tro·flu·o·rom·e·ter \'spek-(ˌ)trō-ˌflü(-ə)r-'äm-ət-ər\ *also* **spec·tro·flu·o·rim·e·ter** \-'im-\ *n* [*spectr-* + *fluorometer*] (1962) : a device for measuring and recording fluorescence spectra — **spec·tro·flu·o·ro·met·ric**

RCL011402

Case 5:03-cv-04669-JW Document 238-5 Filed 08/27/2004 Page 6 of 11

in-ter-fold
in-ter-fra-ter-nal-ty
in-ter-gang
in-ter-gan-gr-a-tion
in-ter-gan-gr-a-tion-al
in-ter-graft
in-ter-gran-u-lar
in-ter-group
in-ter-host
in-ter-hu-mi-vid-u-al
in-ter-in-dus-try
in-ter-in-flu-ence
in-ter-in-sti-tu-tion-al
in-ter-in-volve
in-ter-is-land
in-ter-in-ter-sine
in-ter-lam-i-nar
in-ter-lam-i-nate
in-ter-lam-i-na-tion
in-ter-lay
in-ter-lay-er
in-ter-lay-er-ing
in-ter-lend
in-ter-lev-el
in-ter-li-brary
in-ter-lin-er
in-ter-lob-u-lar
in-ter-lo-cal
in-ter-male
in-ter-mar-gin-al
in-ter-pine-ci-line
in-ter-mem-brane
in-ter-men-stru-al
in-ter-met-ro-pol-i-tan
in-ter-mis-si-ble
in-ter-mi-tot-ic
in-ter-mo-lar
in-ter-mu-ne-um
in-ter-nu-cle-ar
in-ter-nu-cle-on-ic
in-ter-nu-cle-o-tide
in-ter-ob-serv-er
in-ter-ocean
in-ter-oce-an-ic
in-ter-of-fice
in-ter-op-er-a-bil-i-ty
in-ter-op-er-a-ble
in-ter-op-er-a-tive
in-ter-or-bit-al
in-ter-ur-ban
in-ter-pha-lan-ge-al
in-ter-plan-e-tary
in-ter-plu-vi-al
in-ter-pop-u-la-tion
in-ter-pro-fes-sion-al
in-ter-pro-vin-cial
in-ter-pay-chic
in-ter-re-gion-al
in-ter-re-il-lum
in-ter-ra-cial
in-ter-row
in-ter-sam-ple
in-ter-school
in-ter-sec-tion-al
in-ter-seg-ment
in-ter-seg-men-tal
in-ter-sen-so-ry
in-ter-sex
in-ter-sti-tial
in-ter-sti-ti-al-ly
in-ter-stock
in-ter-stor-age
in-ter-strain
in-ter-strand
in-ter-strat-i-fi-ca-tion
in-ter-strat-i-fy
in-ter-sub-sti-tut-abil-i-ty
in-ter-sub-sti-tut-able
in-ter-sys-tem
in-ter-term
in-ter-ter-mi-nal
in-ter-ter-ri-to-ri-al
in-ter-lex-ts-al
in-ter-trans-lat-able
in-ter-tri-al
in-ter-trib-al
in-ter-troop
in-ter-union
in-ter-unit
in-ter-uni-ver-si-ty
in-ter-val-ley
in-ter-ven-tric-u-lar
in-ter-vil-lage
in-ter-vis-i-bil-i-ty
in-ter-vis-i-ble
in-ter-vis-i-ta-tion
in-ter-war
in-ter-work
in-ter-zon-al
in-ter-zone

in-ter-a-bang *var of* INTERROBANG

in-ter-act \ˌint-ə-ˈrakt\ *vi* (1839) : to act upon one another

in-ter-ac-tant \-ˈrak-tənt\ *n* (1949) : one that interacts

in-ter-ac-tion \ˌint-ə-ˈrak-shən\ *n* (1832) : mutual or reciprocal action or influence — **in-ter-ac-tion-al** \-shnəl, -shən-ᵊl\ *adj*

in-ter-ac-tive \-ˈrak-tiv\ *adj* (1832) 1 : mutually or reciprocally active 2 : of, relating to, or being a two-way electronic communication system (as a telephone, cable television, or a computer) that involves a user's orders (as for information or merchandise) or responses (as to a poll) — **in-ter-ac-tive-ly** *adv*

in-ter alia \ˌint-ə-ˈrā-lē-ə, -ˈrī-\ *adv* [L *alia*, neut. pl. of *alius* — more at ELSE] (1665) : among other things

in-ter al-ios \-lē-ˌōs\ *adv* [L *alios*, masc. pl. of *alius*] (1670) : among other persons

in-ter-al-lied \ˌint-ə-ˈral-ˌīd, -rə-ˈlīd\ *adj* (1919) : relating to, composed of, or involving allies

in-ter-atom-ic \ˌint-ə-rə-ˈtäm-ik\ *adj* (1863) : existing or acting between atoms

in-ter-breed \ˌint-ər-ˈbrēd\ *vb* **-bred** \-ˈbred\; **-breed-ing** *vi* (1859) : to breed together: as a : CROSSBREED b : to breed within a closed population ~ *vt* : to cause to breed together

in-ter-ca-la-ry \in-ˈtər-kə-ˌler-ē, ˌint-ər-ˈkal-ə-rē\ *adj* [L *intercalarius*, fr. *intercalare*] (1614) 1 a : inserted in a calendar ⟨an ~ day⟩ b *of a year* : containing an intercalary period (as a day or month) 2 : inserted between other things or parts : INTERPOLATED

in-ter-ca-late \in-ˈtər-kə-ˌlāt\ *vt* **-lat-ed; -lat-ing** [L *intercalatus*, pp. of *intercalare*, fr. *inter-* + *calare* to call, summon — more at LOW] (1614) 1 : to insert (as a day) in a calendar 2 : to insert between or among existing elements or layers *syn* see INTRODUCE — **in-ter-ca-la-tion** \-ˌtər-kə-ˈlā-shən\ *n*

in-ter-cede \ˌint-ər-ˈsēd\ *vi* **-ced-ed; -ced-ing** [L *intercedere*, fr. *inter-* + *cedere* to go — more at CEDE] (1597) : to intervene between parties with a view to reconciling differences : MEDIATE *syn* see INTERPOSE — **in-ter-ced-er** *n*

in-ter-cen-sal \ˌint-ər-ˈsen(t)-səl\ *adj* [*inter-census* + *-al*] (1887) : occurring between censuses ⟨~ estimates⟩ ⟨~ period⟩

¹in-ter-cept \ˌint-ər-ˈsept\ *vt* [L *interceptus*, pp. of *intercipere*, fr. *inter-* + *capere* to take, seize — more at HEAVE] (15c) 1 *obs* : PREVENT; HINDER 2 : to stop, seize, or interrupt in progress or course or before arrival 3 *obs* : to interrupt communication or connection with 4 : to include (part of a curve, surface, or solid) between two points, curves, or surfaces ⟨the part of a circumference ~ed between two radii⟩ 5 : to gain possession of (an opponent's pass)

²in-ter-cept \ˈint-ərˌsept\ *n* (1821) 1 : the distance from the origin to a point where a graph crosses a coordinate axis 2 : INTERCEPTION; *esp* : the interception of a missile by an interceptor or of a target by a missile

in-ter-cept-er \ˌint-ər-ˈsep-tər\ *n* (1601) : INTERCEPTOR

in-ter-cep-tion \ˌint-ər-ˈsep-shən\ *n* (15c) 1 a : the action of intercepting b : the state of being intercepted 2 : something that is intercepted; *esp* : an intercepted forward pass

in-ter-cep-tor \-ˈsep-tər\ *n* (1598) : one that intercepts; *specif* : a light high-speed fast-climbing fighter plane or missile designed for defense against raiding bombers or missiles

in-ter-ces-sion \ˌint-ər-ˈsesh-ən\ *n* [ME or L; ME, fr. L *intercession-*, *intercessio*, fr. *intercessus*, pp. of *intercedere*] (15c) 1 : the act of interceding 2 : prayer, petition, or entreaty in favor of another — **in-ter-ces-sion-al** \-ˈsesh-nəl, -ən-ᵊl\ *adj* — **in-ter-ces-sor** \-ˈses-ər\ *n* — **in-ter-ces-so-ry** \-ˈses-(ə-)rē\ *adj*

¹in-ter-change \ˌint-ər-ˈchānj\ *vb* [ME *entrechaungen*, fr. MF *entrechangier*, fr. OF, fr. *entre-* inter- + *changier* to change] *vt* (14c) 1 : to put each of (two things) in the place of the other 2 : EXCHANGE ~ *vi* : to change places mutually — **in-ter-chang-er** *n*

²in-ter-change \ˈint-ərˌchānj\ *n* (15c) 1 : the act, process, or an instance of interchanging : EXCHANGE 2 : a junction of two or more highways by a system of separate levels that permit traffic to pass from one to another without the crossing of traffic streams

in-ter-change-able \ˌint-ər-ˈchān-jə-bəl\ *adj* (14c) : capable of being interchanged; *esp* : permitting mutual substitution ⟨~ parts⟩ — **in-ter-change-abil-i-ty** \-ˌchān-jə-ˈbil-ət-ē\ *n* — **in-ter-change-able-ness** \-ˈchān-jə-bəl-nəs\ *n* — **in-ter-change-ably** \-blē\ *adv*

in-ter-col-le-giate \ˌint-ər-kə-ˈlē-j(ē-)ət\ *adj* (1873) : existing, carried on, or participating in activities between colleges ⟨~ athletics⟩

in-ter-co-lum-ni-a-tion \ˌint-ər-kə-ˌləm-nē-ˈā-shən\ *n* [L *intercolumnium* space between two columns, fr. *inter-* + *columna* column] (1624) 1 : the clear space between the columns of a series 2 : the system of spacing of the columns of a colonnade

in-ter-com \ˈint-ərˌkäm\ *n* (1940) : INTERCOMMUNICATION SYSTEM

in-ter-com-mu-ni-cate \ˌint-ər-kə-ˈmyü-nəˌkāt\ *vi* (1586) 1 : to exchange communication with one another 2 : to afford passage from one to another — **in-ter-com-mu-ni-ca-tion** \-ˌmyü-nə-ˈkā-shən\ *n*

intercommunication system *n* (1911) : a two-way communication system with microphone and loudspeaker at each station for localized use

in-ter-com-mu-nion \ˌint-ər-kə-ˈmyü-nyən\ *n* (1921) : interdenominational participation in communion

in-ter-con-nect \ˌint-ər-kə-ˈnekt\ *vt* (1865) : to connect with one another ~ *vi* : to be or become mutually connected — **in-ter-con-nec-tion** \-ˈnek-shən\ *n*

in-ter-con-nect-ed *adj* (1865) 1 : mutually joined or related ⟨~ highways⟩ ⟨~ political issues⟩ 2 : having internal connections between the parts or elements ⟨a complex and ~ society⟩ — **in-ter-con-nec-ted-ness** *n*

in-ter-con-ti-nen-tal \ˌint-ər-ˌkänt-ᵊn-ˈent-ᵊl\ *adj* (ca. 1855) 1 : extending among continents or carried on between continents 2 : capable of traveling between continents ⟨~ ballistic missile⟩

in-ter-con-ver-sion \ˌint-ər-kən-ˈvər-zhən, -shən\ *n* (1865) : mutual conversion ⟨~ of chemical compounds⟩ — **in-ter-con-vert** \-ˈvərt\ *vt* — **in-ter-con-vert-ibil-i-ty** \-ˌvərt-ə-ˈbil-ət-ē\ *n* — **in-ter-con-vert-ible** \-ˈvərt-ə-bəl\ *adj*

in-ter-cool-er \ˌint-ər-ˈkü-lər\ *n* (1899) : a device for cooling a fluid (as air) between successive heat-generating processes

in-ter-cos-tal \ˌint-ər-ˈkäs-tᵊl\ *adj* [NL *intercostalis*, fr. L *inter-* + *costa* rib — more at COAST] (1597) : situated or extending between the ribs ⟨~ spaces⟩ ⟨~ muscles⟩ — **intercostal** *n*

in-ter-course \ˈint-ərˌkō(ə)rs, -ˌkȯ(ə)rs\ *n* [ME *intercurse*, prob. fr. MF *entrecours*, fr. ML *intercursus*, fr. L, act of running between, fr. *intercursus*, pp. of *intercurrere* to run between, fr. *inter-* + *currere* to run — more at CURRENT] (15c) 1 : connection or dealings between persons or groups 2 : exchange esp. of thoughts or feelings : COMMUNION 3 : physical sexual contact between individuals that involves the genitalia of at least one person ⟨heterosexual ~⟩ ⟨anal ~⟩ ⟨oral ~⟩; *esp* : SEXUAL INTERCOURSE

in-ter-crop \ˌint-ər-ˈkräp, ˈint-ərˌ\ *vt* (1898) : to grow a crop in between (another) ~ *vi* : to grow two or more crops simultaneously (as in alternate rows) on the same plot — **in-ter-crop** \ˈint-ərˌkräp\ *n*

¹in-ter-cross \ˌint-ər-ˈkrȯs\ *vb* (1711) : CROSS

²in-ter-cross \ˈint-ərˌkrȯs\ *n* (1859) : an instance or a product of crossbreeding

in-ter-cur-rent \ˌint-ər-ˈkər-ənt, -ˈkə-rənt\ *adj* [L *intercurrent-*, *intercurrens*, prp. of *intercurrere*] (1611) : occurring during and modifying the course of another disease ⟨an ~ infection⟩

in-ter-cut \ˌint-ər-ˈkət\ *vt* (1938) 1 : to insert (a contrasting camera shot) into a take by cutting 2 : to insert a contrasting camera shot into (a take) by cutting ~ *vi* : to alternate contrasting camera shots by cutting

in-ter-de-nom-i-na-tion-al \ˌint-ər-di-ˌnäm-ə-ˈnā-shnəl, -shən-ᵊl\ *adj* (1893) : involving or occurring between different denominations — **in-ter-de-nom-i-na-tion-al-ism** \-ˌiz-əm\ *n*

in-ter-den-tal \ˌint-ər-ˈdent-ᵊl\ *adj* (1874) 1 : situated or intended for use between the teeth 2 : formed with the tip of the tongue between the upper and lower front teeth — **in-ter-den-tal-ly** \-ᵊl-ē\ *adv*

in-ter-de-part-men-tal \ˌint-ər-di-ˌpärt-ˈment-ᵊl, -ˌdē-\ *adj* (1895) : carried on between or involving departments (as of an educational institution) — **in-ter-de-part-men-tal-ly** \-ᵊl-ē\ *adv*

¹in-ter-dict \ˈint-ərˌdikt\ *n* [ME *entredit*, fr. OF, fr. L *interdictum* prohibition, praetorian interdict, fr. neut. of *interdictus*, pp. of *interdicere* to interpose, forbid, fr. *inter-* + *dicere* to say — more at DICTION] (13c) 1 : a Roman Catholic ecclesiastical censure withdrawing most sacraments and Christian burial from a person or district 2 : a prohibitory decree : PROHIBITION

²in-ter-dict \ˌint-ərˈdikt\ *vt* (13c) 1 : to lay under or prohibit by an interdict 2 : to forbid in a usu. formal or authoritative manner 3 : to destroy, cut, or damage (as an enemy line of supply) by firepower to stop or hamper an enemy *syn* see FORBID — **in-ter-dic-tion** \-ˈdik-shən\ *n* — **in-ter-dic-tive** \-ˈdik-tiv\ *adj* — **in-ter-dic-tor** \-tər\ *n* — **in-ter-dic-to-ry** \-tə-rē\ *adj*

in-ter-dif-fuse \-dif-ˈyüz\ *vi* (ca. 1860) : to diffuse and mix freely so as to approach a homogeneous mixture — **in-ter-dif-fu-sion** \-ˈyü-zhən\ *n*

in-ter-dig-i-tate \-ˈdij-ə-ˌtāt\ *vi* **-tat-ed; -tat-ing** [*inter-* + L *digitus* finger — more at TOE] (1847) : to become interlocked like the fingers of folded hands — **in-ter-dig-i-ta-tion** \-ˌdij-ə-ˈtā-shən\ *n*

in-ter-dis-ci-pli-nary \-ˈdis-ə-plə-ˌner-ē\ *adj* (1937) : involving two or more academic, scientific, or artistic disciplines

¹in-ter-est \ˈin-trəst; ˈint-ə-ˌrest, -ə-rəst, ˈint-ərst; ˈin-ˌtrest\ *n* [ME, prob. alter. of earlier *interesse*, fr. AF & ML; AF, fr. ML, fr. L, to be between, make a difference, concern, fr. *inter-* + *esse* to be — more at IS] (15c) 1 a (1) : right, title, or legal share in something (2) : participation in advantage and responsibility b : BUSINESS, COMPANY 2 a : a charge for borrowed money generally a percentage of the amount borrowed b : an excess above what is due 3 : ADVANTAGE, BENEFIT; *also* : SELF-INTEREST 4 : SPECIAL INTEREST 5 a : a feeling that accompanies or causes special attention to an object or class of objects : CONCERN b : something that arouses such attention c : a quality in a thing that arouses interest

²interest *vt* (1608) 1 : to induce or persuade to participate or engage 2 : to engage the attention or arouse the interest of

in-ter-est-ed *adj* (1665) 1 : having the attention engaged ⟨~ listeners⟩ 2 : being affected or involved ⟨~ parties⟩ — **in-ter-est-ed-ly** *adv*

RCL011403

Case 5:03-cv-04669-JW Document 238-5 Filed 08/27/2004 Page 7 of 11

[2]request *vt* (1533) 1 : to make a request to or of ⟨*~ed* her to write a paper⟩ 2 : to ask as a favor or privilege ⟨*~s* to be excused⟩ 3 *obs* : to ask (a person) to come or go to a thing or place 4 : to ask for ⟨*~ed* a brief delay⟩ *syn* see ASK — **re-quest-er** *or* **re-quest-or** \-'kwes-tər\ *n*

re-qui-em \'rek-wē-əm *also* 'rāk- *or* 'rēk-\ *n* [ME, fr. L (first word of the introit of the requiem mass), acc. of *requies* rest, fr. *re-* + *quies* quiet, rest — more at WHILE] (14c) 1 : a mass for the dead 2 **a** : a solemn chant (as a dirge) for the repose of the dead **b** : something that resembles such a solemn chant 3 **a** : a musical setting of the mass for the dead **b** : a musical composition in honor of the dead

requiem shark *n* (ca. 1900) : any of a family (Carcharhinidae) of sharks that includes some dangerous to man

re-qui-es-cat \,rek-wē-'es-,kät\ -,at; ,rā-kwē-'es-,kät\ *n* [L, may he (or she) rest, fr. *requiescere* to rest, fr. *re-* + *quiescere* to be quiet, fr. *quies*] (1824) : a prayer for the repose of a dead person

re-quire \ri-'kwī(ə)r\ *vb* **re-quired; re-quir-ing** [ME *requeren*, fr. MF *requerre*, fr. (assumed) VL *requaerere* to seek for, need, require, alter. of L *requirere*, fr. *re-* + *quaerere* to seek, ask] *vt* (14c) 1 **a** : to claim or ask for by right and authority ⟨this night your soul is *required* of you —Lk 12:20 (RSV)⟩ **b** *archaic* : REQUEST 2 **a** : to call for as suitable or appropriate ⟨the occasion *~s* formal dress⟩ **b** : to demand as necessary or essential : have a compelling need for ⟨all living beings *~* food⟩ 3 : to impose a compulsion or command on : COMPEL 4 *chiefly Brit* : to feel or be obliged — used with a following infinitive ⟨one does not *~* to be a specialist —Elizabeth Bowen⟩ *~ vi, archaic* : ASK *syn* see DEMAND

re-quire-ment \-'kwī(ə)r-mənt\ *n* (1662) : something required: **a** : something wanted or needed : NECESSITY ⟨production was not sufficient to satisfy military *~s*⟩ **b** : something essential to the existence or occurrence of something else : CONDITION ⟨failed to meet the school's *~s* for graduation⟩

req-ui-site \'rek-wə-zət\ *adj* [ME, fr. L *requisitus*, pp. of *requirere*] (15c) : ESSENTIAL, NECESSARY — **requisite** *n* — **req-ui-site-ness** *n*

req-ui-si-tion \,rek-wə-'zish-ən\ *n* [MF or ML; MF, fr. ML *requisition-, requisitio*, fr. L, act of searching, fr. *requisitus*, pp.] (1503) 1 **a** : the act of formally requiring or calling upon someone to perform an action **b** : a formal demand made by one nation upon another for the surrender or extradition of a fugitive from justice 2 **a** : the act of requiring something to be furnished **b** : a demand or application made usu. with authority: as (1) : a demand made by military authorities upon civilians for supplies or other needs (2) : a written request for something authorized but not made available automatically 3 : the state of being in demand or use — **requisition** *vt*

re-quit-al \ri-'kwīt-ᵊl\ *n* (1582) 1 : something given in return, compensation, or retaliation 2 : the act or action of requiting : the state of being requited

re-quite \ri-'kwīt\ *vt* **re-quit-ed; re-quit-ing** [*re-* + obs. *quite* (to quit, pay), fr. ME *quiten* — more at QUIT] (1529) 1 **a** : to make return for : REPAY **b** : to make retaliation for : AVENGE 2 : to make suitable return to for a benefit or service or for an injury *syn* see RECIPROCATE, PAY — **re-quit-er** *n*

re-ra-di-ate \(ˌ)rē-'rād-ē-ˌāt\ *vt* (1913) : to radiate again or anew; *esp* : to emit ⟨energy⟩ in the form of radiation after absorbing incident radiation — **re-ra-di-a-tion** \(ˌ)rē-ˌrād-ē-'ā-shən\ *n*

rere-dos \'rer-ə-ˌdäs *also* 'rir-ə-ˌdäs *or* 'ri(ə)r-ˌdäs\ *n* [ME, fr. AF *arere-dos*, fr. MF *arrere* behind + *dos* back, fr. L *dorsum* — more at ARREAR] (15c) 1 : a usu. ornamental wood or stone screen or partition wall behind an altar 2 : the back of a fireplace or open hearth

[1]re-re-lease \,rē-ri-'lēs\ *n* (1945) : something that is rereleased

[2]rerelease *vt* (1967) : to release (as a movie or record) again

rereward *n* [ME *rerewarde*, fr. AF, fr. OF *rere* behind + ONF *warde* guard; akin to OF *garde* guard — more at REAR GUARD] *obs* (14c) : REAR GUARD

[1]re-run \(ˌ)rē-'rən\ *vt* **-ran** \-'ran\; **-run; -run-ning** (1903) : to run again or anew

[2]re-run \'rē-ˌrən, (ˌ)rē-'\ *n* (ca. 1934) : the act or action or an instance of rerunning : REPETITION; *esp* : a presentation of a motion-picture film or television program after its first run

res \'rās, 'rēz\ *n, pl* **res** [L — more at REAL] (1623) : a particular thing : MATTER — used esp. in legal phrases

res ad-ju-di-ca-ta \'rēs-zə-ˌjüd-i-'kät-ə\ *n* [LL] (1902) : RES JUDICATA

re-sal-able \(ˌ)rē-'sā-lə-bəl\ *adj* (1866) : fit for resale

re-sale \'rē-ˌsāl, (ˌ)rē-'sā(ə)l\ *n* (1625) 1 : the act of selling again usu. to a new party 2 **a** : a secondhand sale **b** : an additional sale to the same buyer

re-scale \(ˌ)rē-'skā(ə)l\ *vt* (1944) : to plan, establish, or formulate on a new and usu. smaller scale

re-scind \ri-'sind\ *vt* [L *rescindere* to annul, fr. *re-* + *scindere* to cut — more at SHED] (1643) 1 : to take away : REMOVE 2 **a** : to take back : ANNUL, CANCEL ⟨refused to *~* the order⟩ **b** : to abrogate (a contract) by restoring to the opposite party what one has received from him 3 : to make void (as an act) by action of the enacting authority or a superior authority : REPEAL — **re-scind-er** *n* — **re-scind-ment** \-'sin(d)-mənt\ *n*

re-ci-sion \ri-'sizh-ən\ *n* [LL *rescission-, rescissio*, fr. L *rescissus*, pp. of *rescindere*] (ca. 1611) : an act of rescinding

re-ci-so-ry \-'siz-ə-rē, -'sis-\ *adj* (1605) : relating to or tending to or having the effect of rescission

re-script \'rē-ˌskript\ *n* [L *rescriptum*, fr. neut. of *rescriptus*, pp. of *rescribere* to write in reply, fr. *re-* + *scribere* to write — more at SCRIBE] (1528) 1 : a written answer of a Roman emperor or of a pope to a legal inquiry or petition 2 : an official or authoritative order, decree, edict, or announcement 3 : an act or instance of rewriting

res-cue \'res-(ˌ)kyü\ *vt* **res-cued; res-cu-ing** [ME *rescuen*, fr. MF *rescourre*, fr. OF, fr. *re-* + *escourre* to shake out, fr. L *excutere*, fr. *ex-* + *quatere* to shake — more at QUASH] (14c) : to free from confinement, danger, or evil : SAVE, DELIVER: as **a** : to take (as a prisoner) forcibly from legal custody **b** : to recover (as a prize) by force **c** : to deliver (as a place under siege) by armed force — **res-cu-able** \-ə-bəl\ *adj* — **rescue** *n* — **res-cu-er** *n*

syn RESCUE, DELIVER, REDEEM, RANSOM, RECLAIM, SAVE mean to set free from confinement or danger. RESCUE implies freeing from imminent danger by prompt or vigorous action; DELIVER implies release usu. of a person from confinement, temptation, slavery, or suffering; REDEEM implies releasing from bondage or penalties by giving what is demanded or necessary; RANSOM specif. applies to buying out of captivity; RECLAIM suggests a bringing back to a former state or condition of someone or something abandoned or debased; SAVE may replace any of the foregoing terms; it may further imply a preserving or maintaining for usefulness or continued existence.

rescue mission *n* (1902) : a city religious mission seeking to convert and rehabilitate the down-and-out

[1]re-search \ri-'sərch, 'rē-ˌ\ *n* [MF *recerche*, fr. *recerchier* to investigate thoroughly, fr. OF, fr. *re-* + *cerchier* to search — more at SEARCH] (1577) 1 : careful or diligent search 2 : studious inquiry or examination; *esp* : investigation or experimentation aimed at the discovery and interpretation of facts, revision of accepted theories or laws in the light of new facts, or practical application of such new or revised theories or laws 3 : the collecting of information about a particular subject

[2]research *vt* (1593) 1 : to search or investigate exhaustively ⟨*~* a problem⟩ 2 : to do research for ⟨*~* a book⟩ *~ vi* : to engage in research — **re-search-able** \-ə-bəl\ *adj* — **re-search-er** *n*

re-search-ist \-'sər-chəst, -ˌsər-\ *n* (1921) : one engaged in research

re-seau \rā-'zō, ri-\ *n, pl* **re-seaux** \-'zōz\ [F *réseau*, fr. OF *resel*, dim. of *rais* net, fr. L *retis, rete* — more at RETINA] (1578) 1 : a net ground or foundation in lace 2 : a system of lines forming small squares of standard size photographed by a separate exposure on the same plate with star images to facilitate measurements 3 : a screen with minute elements of three colors in a regular geometric pattern used for taking color photographs

re-sect \ri-'sekt\ *vt* [L *resectus*, pp. of *resecare* to cut off, fr. *re-* + *secare* to cut — more at SAW] (1846) : to perform resection on — **re-sect-abil-i-ty** \-ˌsek-tə-'bil-ət-ē\ *n* — **re-sect-able** \-'sek-tə-bəl\ *adj*

re-sec-tion \ri-'sek-shən\ *n* (1775) : the surgical removal of part of an organ or structure

reseda \'rē-zə-də\ *n* [F *réséda*, fr. *réséda*, a mignonette] (1873) : a variable color averaging a grayish green

re-seed \(ˌ)rē-'sēd\ *vt* (1888) 1 : to sow seed on again or anew 2 : to maintain (itself) by self-sown seed *~ vi* : to maintain itself by self-sown seed

re-sem-blance \ri-'zem-blən(t)s\ *n* (14c) 1 **a** : the quality or state of resembling; *esp* : correspondence in appearance or superficial qualities **b** : a point of likeness : SIMILARITY 2 : REPRESENTATION, IMAGE 3 *archaic* : characteristic appearance 4 *obs* : PROBABILITY *syn* see LIKENESS

re-sem-blant \-blənt\ *adj* (14c) : marked by or showing resemblance

re-sem-ble \ri-'zem-bəl\ *vt* **re-sem-bled; re-sem-bling** \-b(ə-)liŋ\ [ME *resemblen*, fr. MF *resembler, ressembler*, fr. OF, fr. *re-* + *sembler* to be like, seem, fr. L *similare* to copy, fr. *similis* like — more at SAME] (14c) 1 : to be like or similar to 2 *archaic* : to represent as like

re-send \(ˌ)rē-'send\ *vt* **-sent** \-'sent\; **-send-ing** (1554) : to send again or back

re-sent \ri-'zent\ *vt* [F *ressentir* to be emotionally sensible of, fr. OF, fr. *re-* + *sentir* to feel, fr. L *sentire* — more at SENSE] (1628) : to feel or express annoyance or ill will at

re-sent-ful \-fəl\ *adj* (1656) 1 : full of resentment : inclined to resent 2 : caused or marked by resentment — **re-sent-ful-ly** \-fə-lē\ *adv* — **re-sent-ful-ness** *n*

re-sent-ment \ri-'zent-mənt\ *n* (1619) : a feeling of indignant displeasure or persistent ill will at something regarded as a wrong, insult, or injury *syn* see OFFENSE

re-ser-pine \ri-'sər-ˌpēn, -pən\ *n* [G *reserpin*, prob. irreg. fr. NL *Rauwolfia serpentina*, a species of rauwolfia] (1952) : a drug $C_{33}H_{40}N_2O_9$ extracted esp. from the root of rauwolfias and used in the treatment of hypertension, mental disorders, and tension states

res-er-va-tion \,rez-ər-'vā-shən\ *n* (14c) 1 : an act of reserving something: as **a** (1) : the act or fact of a grantor's reserving some newly created thing out of the thing granted (2) : the right or interest so reserved **b** : the setting of limiting conditions or withholding from complete exposition ⟨answered without *~*⟩ **c** : an arrangement to have something (as a hotel room) held for one's use; *also* : a promise, guarantee, or record of such engagement 2 : something reserved: as **a** : a tract of public land set aside (as for the use of American Indians) **b** : an area in which hunting is not permitted; *esp* : one set aside as a secure breeding place 3 **a** : a limiting condition ⟨agreed, but with *~s*⟩ **b** : a specific objection ⟨had *~s* about the finding⟩

[1]re-serve \ri-'zərv\ *vt* **re-served; re-serv-ing** [ME *reserven*, fr. MF *reserver*, fr. L *reservare*, lit., to keep back, fr. *re-* + *servare* to keep — more at CONSERVE] (14c) 1 **a** : to hold in reserve : keep back ⟨*~* grain for seed⟩ **b** : to set aside (part of the consecrated elements) at the Eucharist for future use **c** : to retain or hold over to a future time or place : DEFER ⟨*~* one's judgment on a plan⟩ **d** : to make legal reservation of 2 : to set or have set aside or apart ⟨*~* a hotel room⟩ *syn* see KEEP

[2]reserve *n, often attrib* (1648) 1 : something stored or kept available for future use or need : STOCK 2 : something reserved or set aside for a particular purpose, use, or reason: as **a** (1) : a military force withheld from action for later decisive use — usu. used in pl. (2) : forces not in the field but available (3) : the military forces of a country not part of the regular services; *also* : RESERVIST **b** : a tract (as of public land) set apart : RESERVATION 3 : an act of reserving : QUALIFICATION 4 **a** : restraint, closeness, or caution in one's words and actions **b** : forbearance from making a full explanation, complete disclosure, or free expression of one's mind 5 *archaic* : SECRET 6 **a** : money or its equivalent kept in hand or set apart usu. to meet liabilities **b** : the liquid resources of a nation for meeting international payments 7 : SUBSTITUTE 8 : RESERVE PRICE — **in reserve** : held back for future or special use

reserve bank *n* (1905) : a central bank holding reserves of other banks

reserve clause *n* (1944) : the clause in a professional athlete's contract that reserves for the club the exclusive right automatically to renew the contract and that binds the athlete to the club for his entire playing career or until he is traded or released

re-served \ri-'zərvd\ *adj* (1601) 1 : restrained in words and actions 2 : kept or set apart or aside for future or special use *syn* see SILENT — **re-serv-ed-ly** \-'zər-vəd-lē\ *adv* — **re-serv-ed-ness** \-'zər-vəd-nəs\ *n*

reserved power *n* (1835) : a political power reserved by a constitution to the exclusive jurisdiction of a specified political authority

RCL011404

Case 5:03-cv-04669-JW Document 238-5 Filed 08/27/2004 Page 8 of 11

Ruhm·korff coil \ˌrüm-ˌkȯrf-, ˌrüm-\ *n* [Heinrich *Ruhmkorff* †1877 Ger. physicist] (1852) : INDUCTION COIL

¹ru·in \ˈrü-ən, -ˌin\ *n* [ME *ruine*, fr. MF, fr. L *ruina*; akin to L *ruere* to fall — more at RUG] (14c) 1 *archaic* : a falling down : COLLAPSE ⟨from age to age . . . the crash of ~ fitfully resounds —William Wordsworth⟩ b : physical, moral, economic, or social collapse 2 a : the state of being ruined ⟨the city lay in ~s⟩ b : the remains of something destroyed — usu. used in pl. ⟨the ~s of the ancient world —William Hazlitt †1830⟩ 3 : a cause of destruction 4 a : the action of destroying, laying waste, or wrecking b : DAMAGE, INJURY 5 : a ruined building, person, or object — **ru·in·ate** \-ə-ˌnət, -ˌnāt\ *adj* — **ru·in·ate** \-ˌnāt\ *vt*

²ruin *vt* (1581) 1 : to reduce to ruins : DEVASTATE 2 a : to damage irreparably b : BANKRUPT, IMPOVERISH ⟨~ed by speculation⟩ 3 : to subject to frustration, failure, or disaster ⟨will ~ your chances of promotion⟩ ~ *vi* : to become ruined — **ru·in·er** *n*

ru·in·ation \ˌrü-ə-ˈnā-shən\ *n* (1664) : RUIN, DESTRUCTION

ru·in·ous \ˈrü-ə-nəs\ *adj* (14c) 1 : DILAPIDATED, RUINED 2 : causing or tending to cause ruin — **ru·in·ous·ly** *adv* — **ru·in·ous·ness** *n*

¹rule \ˈrül\ *n* [ME *reule*, fr. OF, fr. L *regula* straightedge, rule, fr. *regere* to lead straight — more at RIGHT] (13c) 1 a : a prescribed guide for conduct or action b : the laws or regulations prescribed by the founder of a religious order for observance by its members c : an accepted procedure, custom, or habit d (1) : a usu. written order or direction made by a court regulating court practice or the action of parties (2) : a legal precept or doctrine e : a regulation or bylaw governing procedure or controlling conduct 2 a (1) : a usu. valid generalization (2) : a generally prevailing quality, state, or mode ⟨fair weather was the ~ yesterday —*N.Y. Times*⟩ b : a standard of judgment : CRITERION c : a regulating principle d : a determinate method for performing a mathematical operation and obtaining a certain result 3 a : the exercise of authority or control : DOMINION b : a period during which a specified ruler or government exercises control 4 a : a strip of material marked off in units used for measuring or ruling off lengths b : a metal strip with a type-high face that prints a linear design; *also* : a linear design produced by or as if by such a strip *syn* see LAW — **as a rule** : for the most part : GENERALLY

²rule *vb* **ruled; rul·ing** *vt* (13c) 1 a : to exert control, direction, or influence on ⟨the superstitions that ~ primitive minds⟩ b : to exercise control over esp. by curbing or restraining ⟨~ a fractious horse⟩ ⟨*ruled* his appetites firmly⟩ 2 a : to exercise authority or power over often harshly or arbitrarily ⟨the speaker *ruled* the legislature with an iron hand⟩ b : to be preeminent in : DOMINATE 3 : to determine and declare authoritatively; *esp* : to command or determine judicially 4 a (1) : to mark with lines drawn along or as if along the straight edge of a ruler (2) : to mark (a line) on a paper with a ruler b : to arrange in a line ~ *vi* 1 a : to exercise supreme authority b : to be first in importance or prominence : PREDOMINATE ⟨the physical did not ~ in her nature —Sherwood Anderson⟩ 2 : to exist in a specified state or condition 3 : to lay down a legal rule *syn* see GOVERN, DECIDE

ruled surface *n* (ca. 1862) : a surface generated by a moving straight line with the result that through every point on the surface a line can be drawn lying wholly in the surface

rule·less \ˈrül-ləs\ *adj* (15c) : not restrained or regulated by law

rule of the road (1871) : a customary practice (as driving always on a particular side of the road or yielding the right of way) developed in the interest of safety and often subsequently reinforced by law; *esp* : any of the rules making up a code governing ships in matters relating to mutual safety

rule of thumb (1692) 1 : a method of procedure based on experience and common sense 2 : a general principle regarded as roughly correct but not intended to be scientifically accurate

rule out *vt* (1869) 1 : EXCLUDE, ELIMINATE 2 : to make impossible : PREVENT ⟨heavy rain *ruled out* the picnic⟩

rul·er \ˈrü-lər\ *n* (14c) 1 : one that rules; *specif* : SOVEREIGN 2 : a worker or a machine that rules paper 3 : a smooth-edged strip (as of wood or metal) that is usu. marked off in units (as inches) and is used as a straightedge or for measuring — **rul·er·ship** \-ˌship\ *n*

¹rul·ing \ˈrü-liŋ\ *n* (1560) : an official or authoritative decision, decree, statement, or interpretation (as by a judge on a point of law)

²ruling *adj* (1593) 1 a : exerting power or authority ⟨the ~ party⟩ b : CHIEF, PREDOMINATING ⟨a ~ passion⟩ 2 : generally prevailing

¹rum \ˈrəm\ *n* [prob. short for obs. *rumbullion* (rum)] (1654) 1 : an alcoholic beverage distilled from a fermented cane product (as molasses) 2 : alcoholic liquor ⟨the demon ~⟩

²rum *adj* **rum·mer; rum·mest** [earlier *rome*, perh. fr. Romany *rom* Gypsy man] (1752) 1 *chiefly Brit* : QUEER, ODD ⟨writing is a ~ trade . . . and what is all right one day is all wrong the next —Angela Thirkell⟩ 2 *chiefly Brit* : DIFFICULT, DANGEROUS

ru·ma·ki \rə-ˈmäk-ē\ *n* (origin unknown) (1961) : a cooked appetizer made of pieces of usu. marinated chicken liver wrapped together with sliced water chestnuts in bacon slices

Ru·ma·nian \rü-ˈmā-nē-ən, -nyən\ *n* (1868) 1 : ROMANIAN 1 2 : the Romance language of the Romanians — **Rumanian** *adj*

rum·ba \ˈrəm-bə, ˈrüm-, ˈrüm-\ *n* [AmerSp] (1922) : a ballroom dance of Cuban origin in 2/4 or 4/4 time with a basic pattern of step-close-step and marked by a delayed transfer of weight and pronounced hip movements; *also* : the music for this dance

¹rum·ble \ˈrəm-bəl\ *vb* **rum·bled; rum·bling** \-b(ə-)liŋ\ [ME *rumblen*; akin to MHG *rummeln* to rumble] *vi* (14c) 1 : to make a low heavy rolling sound ⟨thunder *rumbling* in the distance⟩ 2 : to travel with a low reverberating sound ⟨wagons *rumbled* into town⟩ 3 : to speak in a low rolling tone 4 : to engage in a rumble ~ *vt* 1 : to utter or emit in a low rolling voice 2 : to polish or otherwise treat (metal parts) in a tumbling barrel — **rum·bler** \-b(ə-)lər\ *n*

²rumble *n* (14c) 1 a : a low heavy continuous reverberating often muffled sound (as of thunder) b : low frequency noise in phonographic playback caused by the transmission of mechanical vibrations by the turntable to the pickup 2 : a seat for servants behind the body of a carriage 3 : TUMBLING BARREL 4 a : widespread expression of dissatisfaction or unrest b : a street fight esp. among gangs

rumble seat *n* (1912) : a folding seat in the back of an automobile (as a coupe or roadster) not covered by the top

rum·bling \ˈrəm-bliŋ\ *n* (14c) 1 : RUMBLE 2 : general but unofficial talk or opinion often of dissatisfaction — usu. used in pl. ⟨occasional ~s about . . . government spending —Paul Potter⟩

rum·bly \ˈrəm-b(ə-)lē\ *adj* (1874) : tending to rumble or rattle

rum·bus·tious \ˌrəm-ˈbəs-chəs\ *adj* [alter. of *robustious*] *chiefly Brit* (1778) : RAMBUNCTIOUS — **rum·bus·tious·ly** *adv*, *chiefly Brit* — **rum·bus·tious·ness** *n*, *chiefly Brit*

ru·men \ˈrü-mən\ *n, pl* **ru·mi·na** \-mə-nə\ *or* **rumens** [NL *rumin-, rumen*, fr. L, gullet] (1728) : the large first compartment of the stomach of a ruminant in which cellulose is broken down by the action of symbionts — **ru·mi·nal** \-mən-ᵊl\ *adj*

¹ru·mi·nant \ˈrü-mə-nənt\ *n* (1661) : a ruminant mammal

²ruminant *adj* (ca. 1679) 1 a (1) : chewing the cud (2) : characterized by chewing again what has been swallowed b : of or relating to a suborder (Ruminantia) of even-toed hoofed mammals (as sheep, giraffes, deer, and camels) that chew the cud and have a complex 3- or 4-chambered stomach 2 : given to or engaged in contemplation : MEDITATIVE ⟨stood there with her hands clasped in this attitude of ~ relish —Thomas Wolfe⟩ — **ru·mi·nant·ly** *adv*

ru·mi·nate \ˈrü-mə-ˌnāt\ *vb* **-nat·ed; -nat·ing** [L *ruminatus*, pp. of *ruminari* to chew the cud, muse upon, fr. *rumin-, rumen* gullet; akin to Skt *romantha* ruminant] *vt* (1533) 1 : to go over in the mind repeatedly and often casually or slowly 2 : to chew repeatedly for an extended period ~ *vi* 1 : to chew again what has been chewed slightly and swallowed : chew the cud 2 : to engage in contemplation : REFLECT *syn* see PONDER — **ru·mi·na·tion** \ˌrü-mə-ˈnā-shən\ *n* — **ru·mi·na·tive** \ˈrü-mə-ˌnāt-iv\ *adj* — **ru·mi·na·tive·ly** *adv* — **ru·mi·na·tor** \-ˌnāt-ər\ *n*

¹rum·mage \ˈrəm-ij\ *vb* **rum·maged; rum·mag·ing** *vi* (1595) 1 : to make a thorough search or investigation 2 : to engage in an undirected or haphazard search ~ *vt* 1 : to make a thorough search through : RANSACK ⟨*rummaged* the attic⟩ 2 : to examine minutely and completely 3 : to discover by searching — **rum·mag·er** *n*

²rummage *n* [obs. E *rummage* act of packing cargo, modif. of MF *arrimage*] (1598) 1 a : a confused miscellaneous collection b : items for sale at a rummage sale 2 : a thorough search esp. among a confusion of objects

rummage sale *n* (ca. 1858) : a usu. informal sale of miscellaneous goods; *esp* : a sale of donated articles conducted by a nonprofit organization (as a church or charity) to help support its programs

rum·mer \ˈrəm-ər\ *n* [G or D; G *römer*, fr. D *roemer*] (1654) : a large-bowled footed drinking glass often elaborately etched or engraved

¹rum·my \ˈrəm-ē\ *adj* **rum·mi·er; -est** (1828) : QUEER, ODD ⟨were still feeling a little ~ from our trip up the escalator —*New Yorker*⟩

²rummy *n, pl* **rummies** (1851) : DRUNKARD

³rummy *n* [perh. fr. *rummy*] (1915) : any of several card games for two or more players in which each player tries to assemble groups of three or more cards of the same rank or suit and to be the first to meld all his cards

¹ru·mor \ˈrü-mər\ *n* [ME *rumour*, fr. MF, fr. L *rumor*; akin to OE *rēon* to lament, Gk *ōryesthai* to howl] (14c) 1 : talk or opinion widely disseminated with no discernible source 2 : a statement or report current without known authority for its truth 3 *archaic* : talk or report of a notable person or event 4 : a soft low indistinct sound : MURMUR

²rumor *vt* **ru·mored; ru·mor·ing** \ˈrüm-(ə-)riŋ\ (1594) : to tell or spread by rumor

ru·mor·mon·ger \-ˌmən-gər, -ˌmäŋ-\ *n* (1917) : one who spreads rumors

ru·mour \ˈrü-mər\ *chiefly Brit var of* RUMOR

rump \ˈrəmp\ *n* [ME, of Scand origin; akin to Icel *rumpr* rump; akin to MHG *rumph* torso] (15c) 1 a : the upper rounded part of the hindquarters of a quadruped mammal b : BUTTOCKS c : the sacral or dorsal part of the posterior end of a bird 2 : a cut of beef between the loin and round — see BEEF illustration 3 : a small fragment remaining after the separation of the larger part of a group or an area; *esp* : a group (as a parliament) carrying on in the name of the original body after the departure or expulsion of a large number of its members

¹rum·ple \ˈrəm-pəl\ *n* (ca. 1500) : FOLD, WRINKLE

²rumple *vb* **rum·pled; rum·pling** \-p(ə-)liŋ\ [D *rompelen*; akin to OHG *rimpfan* to wrinkle, L *curvus* curved — more at CROWN] *vt* (1603) 1 : WRINKLE, CRUMPLE 2 : to make unkempt : TOUSLE ~ *vi* : to become rumpled

rum·ply \ˈrəm-p(ə-)lē\ *adj* **rum·pli·er; -est** (1833) : having rumples

rum·pus \ˈrəm-pəs\ *n* [origin unknown] (1764) : a usu. noisy commotion

rumpus room *n* (1939) : a room usu. in the basement of a home that is used for games, parties, and recreation

rum-run·ner \ˈrəm-ˌrən-ər\ *n* (1920) : a person or ship engaged in bringing prohibited liquor ashore or across a border — **rum-run·ning** \-ˌrən-iŋ\ *adj or n*

¹run \ˈrən\ *vb* **ran** \ˈran\; **run; run·ning** [ME *rönnen*, alter. of *rinnen*, v.i. (fr. OE *iernan, rinnan* & ON *rinna*) & of *rennen*, v.t., fr. ON *renna*; akin to OHG *rinnan*, v.i., to run, OE *risan* to rise] *vi* (bef. 12c) 1 a : to go faster than a walk; *specif* : to go steadily by springing steps so that both feet leave the ground for an instant in each step b *of a horse* : to move at a fast gallop c : FLEE, RETREAT, ESCAPE ⟨dropped the gun and *ran*⟩ d : to utilize a running play on offense — used of a football team 2 a : to go without restraint : move freely about at will ⟨let chickens ~ loose⟩ b : to keep company : CONSORT ⟨a ram *running* with ewes⟩ ⟨*ran* with a wild crowd when he was young⟩ c : to sail before the wind in distinction from reaching or sailing close-hauled d : ROAM, ROVE ⟨*running* about with no overcoat⟩ 3 a : to go rapidly or hurriedly : HASTEN ⟨~ and fetch the doctor⟩ b : to go in urgency or distress : RESORT ⟨~s to mother at every little difficulty⟩ c : to make a quick, easy, or casual trip or visit ⟨*ran* over to borrow some sugar⟩ 4 a : to contend in a race b : to enter into an election contest 5 a : to move on or as if on wheels : GLIDE ⟨file drawers *running* on ball bearings⟩ b : to roll forward rapidly or freely c : to pass or slide freely ⟨a rope ~s through the pulley⟩ d : to ravel lengthwise ⟨stockings guaranteed not to ~⟩ 6 : to sing or play a musical passage quickly ⟨~ up the scale⟩ 7 a : to go back and forth : PLY ⟨the train ~s between New York and Washington⟩ b *of fish* : to migrate or move in schools; *esp* : to ascend a river to spawn 8 a : TURN, ROTATE ⟨a swiftly *running* grindstone⟩ b : FUNCTION, OPERATE ⟨the engine ~s on gasoline⟩ 9 a : to continue in force or operation ⟨the contract has two more

years to ~⟩ b : to
of-way that ~s wit
payable ⟨interest or
state to another ⟨~
sure b : MELT, FUSE
d : to discharge pu
rapidly in some sp
shoot of growth b
or feature ⟨they ~ t
a certain direction ⟨
relation to somethir
tain form or expres
tain order of succes
~s in the family⟩ t
or quality ⟨profits w
uous range of varia
play on a stage a nu
for six months⟩ 15
⟨chills *ran* up her s
ran rife⟩ ~ *vt* 1 a
fast b : to bring to
himself to death⟩ c
deer⟩ d : to follow
source⟩ e : to enter
put forward as a can
a grazing place b :
maintain (livestock)
traverse with speed
ning ⟨*ran* a great r
through or past ⟨~
penetrate or enter :
: to cause to pass : t
to collide ⟨*ran* his h
cause to pass lightly
eye down the list⟩ 6
go in a specified man
ERATE ⟨~ a lathe⟩ c
CONDUCT ⟨~ a factor
ran blood⟩ b : CONT
fied way or into a sp
melt and cast in a mo
in a still⟩ ⟨~ a prob
liable to : INCUR ⟨*ran*
⟨~ a contour line on
mulate before settlin
⟨rooms that ~ $50 a
— usu. used with *off*
carry in a printed me
a : to make (a series
billiards⟩ b : to lea
make (a golf ball) rol
with or discover by c
seek the company of
theories⟩ — **run again**
work or take effect un
ature : to have a feve
rectly for the game ins
1 : to collide with ⟨*ra*
with ⟨*run foul of* the l
into : BECOME b : to
income often *runs into*
TER, MEET ⟨*ran into* an
: to show marked su
ingly — **run riot** 1 :
profusion — **run short**
up — **run to** : to mour
: to run across : meet

²run *n* (15c) 1 a : an
movement b : a quic
ascending a river to sp
ascend a river to spaw
made in baseball by a
or ability to run g : a
ning play in football ⟨
: CREEK 2 b : someth
tion or during a certain
: the stern of the unde
curve or slope upward
ore lies c : a direction
of a mass of granite⟩
flight of steps (2) : th
center line of a buildin
tinuous series esp. of th
passage up or down a s
ber of rapid small danc
making successively a r
score thus made ⟨a ~
performances or showi
readings, or observatic
depositors, creditors, o
5 : the quantity of worl
~ of 10,000 copies⟩ 6
group ⟨the average ~
period of continuous tr
mapped out and travele
regular territory : BEAT
ing the ground e : fre
area ⟨has the ~ of the
chine or plant is in cont
a single set of processin
track, or path frequente
animals where they may
of land used for grazin
holder⟩ d : an inclined

RCL011405

Case 5:03-cv-04669-JW Document 238-5 Filed 08/27/2004 Page 9 of 11

con-clu-sion \kən-'klü-zhən\ *n* [ME, fr. MF, fr. L *conclusion-, conclusio,* fr. *conclusus,* pp. of *concludere*] (14c) **1** : a reasoned judgment : INFERENCE **b** : the necessary consequence of two or more propositions taken as premises; *esp* : the inferred proposition of a syllogism **2** : the last part of something; as **a** : RESULT, OUTCOME **b** *pl* : trial of strength or skill — used in the phrase *try conclusions* **c** : a final summation **d** : the final decision in a law case **e** : the final part of a pleading in law **3** : an act or instance of concluding

con-clu-sive \-'klü-siv, -ziv\ *adj* (1612) **1** : of or relating to a conclusion **2** : putting an end to debate or question esp. by reason of irrefutability — **con-clu-sive-ly** *adv* — **con-clu-sive-ness** *n*

syn CONCLUSIVE, DECISIVE, DETERMINATIVE, DEFINITIVE mean bringing to an end. CONCLUSIVE applies to reasoning or logical proof that puts an end to debate or questioning; DECISIVE may apply to something that ends a controversy, a contest, or any uncertainty; DETERMINATIVE adds an implication of giving a fixed course or direction; DEFINITIVE applies to what is put forth as final and permanent

con-coct \kən-'käkt; kän-\ *vt* [L *concoctus,* pp. of *concoquere* to cook together, fr. *com-* + *coquere* to cook — more at COOK] (1675) **1** : to prepare by combining crude materials **2** : DEVISE, FABRICATE — **con-coct-er** *n* — **con-coc-tion** \-'käk-shən\ *n* — **con-coc-tive** \-'käk-tiv\ *adj*

con-com-i-tance \kən-'käm-ət-ən(t)s, kän-\ *n* (1535) : ACCOMPANIMENT; *esp* : a conjunction that is regular and is marked by correlative variation of accompanying elements

con-com-i-tant \-ət-ənt\ *adj* [L *concomitant-, concomitans,* prp. of *concomitari* to accompany, fr. *com-* + *comitari* to accompany, fr. *comit-, comes* companion — more at COUNT] (1607) : accompanying esp. in a subordinate or incidental way — **con-com-i-tant-ly** *adv*

²concomitant *n* (1621) : something that accompanies or is collaterally connected with something else : ACCOMPANIMENT

con-cord \'kän-ˌkȯ(ə)rd, 'kaŋ-\ *n* [ME, fr. MF *concorde,* fr. L *concordia,* fr. *concord-, concors* agreeing, fr. *com-* + *cord-, cor* heart — more at HEART] (14c) **1** **a** : a state of agreement : HARMONY **b** : a simultaneous occurrence of two or more musical tones that produces an impression of agreeableness or resolution on a listener — compare DISCORD **2** : agreement by stipulation, compact, or covenant **3** : grammatical agreement

con-cor-dance \kən-'kȯrd-ᵊn(t)s, kän-\ *n* [ME, fr. MF, fr. ML *concordantia,* fr. L *concordant-, concordans,* prp. of *concordare* to agree, fr. *concord-, concors*] (14c) **1** : an alphabetical index of the principal words in a book or the works of an author with their immediate contexts **2** : CONCORD, AGREEMENT

con-cor-dant \-ᵊnt\ *adj* [ME, fr. MF, fr. L *concordant-, concordans*] (15c) : CONSONANT, AGREEING — **con-cor-dant-ly** *adv*

con-cor-dat \kän-'kȯr-ˌdat\ *n* [F, fr. ML *concordatum,* fr. L, neut. of *concordatus,* pp. of *concordare*] (1616) : COMPACT, COVENANT; *specif* : an agreement between a pope and a sovereign or government for the regulation of ecclesiastical matters

con-cours d'e-le-gance \(ˌ)kōⁿ-ˌkü(ə)r-dā-lā-'gäⁿs\ *n* [F *concours d'élégance,* lit., competition of elegance] (1950) : a show or contest of vehicles and accessories in which the entries are judged chiefly on excellence of appearance and turnout

con-course \'kän-ˌkȯ(ə)rs, 'käŋ-, -kȯ(ə)rs\ *n* [ME, fr. MF & L; MF *concours,* fr. L *concursus,* fr. *concursus,* pp. of *concurrere* to run together — more at CONCUR] (14c) **1** : an act or process of coming together and merging **2** : a meeting produced by voluntary or spontaneous coming together **3** **a** : an open space where roads or paths meet **b** : an open space or hall (as in a railroad terminal) where crowds gather

con-cres-cence \kən-'kres-ᵊn(t)s, kän-\ *n* [L *concrescentia,* fr. *concrescent-, concrescens,* prp. of *concrescere* to grow together, fr. *com-* + *crescere* to grow — more at CRESCENT] (1614) **1** : increase by the addition of particles **2** : a growing together : COALESCENCE; *esp* : convergence and fusion of the lateral lips of the blastopore to form the primordium of an embryo — **con-cres-cent** \-ᵊnt\ *adj*

¹con-crete \(ˌ)kän-'krēt, 'kän-ˌ, kən-\ *adj* [ME, fr. L *concretus,* fr. pp. of *concrescere*] (14c) **1** : naming a real thing or class of things ⟨the word *poem* is ~, *poetry* is abstract⟩ **2** : formed by coalition of particles into one solid mass **3** **a** : characterized by or belonging to immediate experience of actual things or events **b** : SPECIFIC, PARTICULAR **c** : REAL, TANGIBLE **4** : relating to or made of concrete — **con-crete-ly** *adv* — **con-crete-ness** *n*

²con-crete \'kän-ˌkrēt, kän-\ *vb* **con-cret-ed; con-cret-ing** *vt* (1635) **1** a : to form into a solid mass : SOLIDIFY **b** : COMBINE, BLEND **2** : to make actual or real : cause to take on the qualities of reality **3** : to cover with, form of, or set in concrete ~ *vi* : to become concreted

³con-crete \'kän-ˌkrēt, (ˌ)kän-\ *n* (1656) **1** : a mass formed by coneretion or coalescence of separate particles of matter in one body **2** : a hard strong building material made by mixing a cementing material (as portland cement) and a mineral aggregate (as sand and gravel) with sufficient water to cause the cement to set and bind the entire mass **3** : a waxy essence of flowers prepared by extraction and evaporation and used in perfumery

concrete music *n* (1950) : MUSIQUE CONCRÈTE

concrete poetry *n* (1958) : poetry in which the poet's intent is conveyed by the graphic patterns of letters, words, or symbols rather than by the conventional arrangement of words

con-cre-tion \kän-'krē-shən, kən-\ *n* (1514) **1** : something concreted: as **a** : a hard usu. inorganic mass (as a bezoar or tophus) formed in a living body **b** : a mass of mineral matter found generally in rock of a composition different from its own and produced by deposition from aqueous solution in the rock **2** : the act or process of concreting : the state of being concreted ⟨~ of ideas in an hypothesis⟩ — **con-cre-tion-ary** \-shə-ˌner-ē\ *adj*

con-cret-ism \kän-'krē-ˌti-zəm, 'kän-ˌ\ *n* (1865) : representation of abstract things as concrete; *esp* : the theory or practice of concrete poetry — **con-cret-ist** \-'krē-ˌtist, -ˌkrēt-\ *n*

con-cre-tize \-ˌtīz\ *vb* **-tized; -tiz-ing** *vt* (1884) : to make concrete, specific, or definite (tried to ~ his ideas) ~ *vi* : to become concrete — **con-cre-ti-za-tion** \(ˌ)kän-ˌkrēt-ə-'zā-shən\ *n*

con-cu-bi-nage \kän-'kyü-bə-nij, kən-\ *n* (14c) **1** : cohabitation of persons not legally married **2** : the state of being a concubine

con-cu-bine \'kän-kyü-ˌbīn, 'kän-\ *n* [ME, fr. MF, fr. L *concubina,* fr. *concubare* to lie — more at HIP] (14c) **1** : a woman living in a socially recognized state of concubinage **2** : MISTRESS

con-cu-pis-cence \'kän-'kyü-pə-sən(t)s, kən-\ *n* [ME, fr. MF, fr. LL *concupiscentia,* fr. L *concupiscent-, concupiscens,* prp. of *concupiscere* to desire ardently, fr. *com-* + *cupere* to desire — more at COVET] (14c) : strong desire; *esp* : sexual desire — **con-cu-pis-cent** \-sənt\ *adj*

con-cu-pis-ci-ble \-'kyü-pə-sə-bəl\ *adj* [ME, fr. MF or LL; MF, fr. LL *concupiscibilis,* fr. L *concupiscere*] (14c) : LUSTFUL, DESIROUS

con-cur \kən-'kər, kän-\ *vi* **con-curred; con-cur-ring** [ME *concurren,* fr. L *concurrere,* fr. *com-* + *currere* to run — more at CAR] (15c) **1** : to act together to a common end or single effect **2** **a** : APPROVE (~ in a statement⟩ **b** : to express agreement ⟨~ with an opinion⟩ **3** *obs* : to come together : MEET **4** : to happen together : COINCIDE *syn* see AGREE

con-cur-rence \-'kər-ən(t)s, -'kə-rən(t)s\ *n* (15c) **1** **a** : agreement or union in action : COOPERATION **b** (1) : agreement in opinion or design (2) : CONSENT **2** : a coming together : CONJUNCTION **3** : a coincidence of equal powers in law

con-cur-rent \-'kər-ənt, -'kə-rənt\ *adj* [ME, fr. MF & L; MF, fr. L *concurrent-, concurrens,* prp. of *concurrere*] (14c) **1** : operating or occurring at the same time **2** **a** : CONVERGENT; *specif* : meeting or intersecting in a point **b** : running parallel **3** : acting in conjunction **4** : exercised over the same matter or area by two different authorities ⟨~ jurisdiction⟩ — **concurrent** *n* — **con-cur-rent-ly** *adv*

concurrent resolution *n* (1802) : a resolution passed by both houses of a legislative body that lacks the force of law

con-cuss \kən-'kəs\ *vt* [L *concussus,* pp.] (1597) : to affect with or as if with concussion

con-cus-sion \kən-'kəsh-ən\ *n* [MF or L; MF, fr. L *concussion-, concussio,* fr. *concussus,* pp. of *concutere,* to shake violently, fr. *com-* + *quatere* to shake] (15c) **1** : AGITATION, SHAKING **2** **a** : a hard blow or collision **b** : a stunning, damaging, or shattering effect from a hard blow; *esp* : a jarring injury of the brain resulting in disturbance of cerebral function — **con-cus-sive** \-'kəs-iv\ *adj*

con-demn \kən-'dem\ *vt* [ME *condemnen,* fr. MF *condemner,* fr. L *condemnare,* fr. *com-* + *damnare* to condemn — more at DAMN] (14c) **1** : to declare to be reprehensible, wrong, or evil usu. after weighing evidence and without reservation **2** **a** : to pronounce guilty : CONVICT **b** : SENTENCE, DOOM **3** : to adjudge unfit for use or consumption **4** : to declare convertible to public use under the right of eminent domain *syn* see CRITICIZE — **con-dem-nable** \-'dem-(n)ə-bəl\ *adj* — **con-dem-na-to-ry** \-nə-ˌtōr-ē, -ˌtȯr-\ *adj* — **con-dem-ner** \-'dem-ər\ *or* **con-dem-nor** \kən-'dem-ər; kən-ˌdem-'nȯ(ə)r, kän-\ *n*

con-dem-na-tion \ˌkän-ˌdem-'nā-shən, -ˌdəm-\ *n* (14c) **1** : CENSURE, BLAME **2** : the act of judicially condemning **3** : the state of being condemned **4** : a reason for condemning

con-den-sate \'kän-dən-ˌsāt, -ˌden-; kən-'den-\ *n* (ca. 1885) : a product of condensation; *esp* : a liquid obtained by condensation of a gas or vapor ⟨steam ~⟩

con-den-sa-tion \ˌkän-ˌden-'sā-shən, -ˌdən-\ *n* (1603) **1** : the act or process of condensing: as **a** : a chemical reaction involving union between molecules often with elimination of a simple molecule (as water) to form a new more complex compound of often greater molecular weight **b** : a reduction to a denser form (as from steam to water) **c** : compression of a written or spoken work into more concise form **2** : the quality or state of being condensed **3** : a product of condensing; *esp* : an abridgment of a literary work — **con-den-sa-tion-al** \-shnəl, -shən-ᵊl\ *adj*

con-dense \kən-'den(t)s\ *vb* **con-densed; con-dens-ing** [ME *condensen,* fr. MF *condenser,* fr. L *condensare,* fr. *com-* + *densare* to make dense, fr. *densus* dense] *vt* (15c) **1** : to make denser or more compact; *esp* : to subject to condensation ~ *vi* : to undergo condensation *syn* see CONTRACT — **con-dens-able** *also* **con-dens-ible** \-'den(t)-sə-bəl\ *adj*

con-densed *adj* (15c) : reduced to a more compact form; *esp* : having a face that is narrower than that of a typeface not so characterized

condensed milk *n* (1863) : evaporated milk with sugar added

con-dens-er \kən-'den(t)-sər\ *n* (1686) **1** : one that condenses: as **a** : a lens or mirror used to concentrate light on an object **b** : an apparatus in which gas or vapor is condensed **2** : CAPACITOR

con-de-scend \ˌkän-di-'send\ *vi* [ME *condescenden,* fr. MF *condescendre,* fr. LL *condescendere,* fr. L *com-* + *descendere* to descend] (14c) **1** **a** : to descend to a less formal or dignified level : UNBEND **b** : to waive the privileges of rank **2** : to assume an air of superiority

con-de-scen-dence \-'sen-dən(t)s\ *n* (1638) : CONDESCENSION

con-de-scend-ing *adj* (1707) : showing or characterized by condescension : PATRONIZING — **con-de-scend-ing-ly** *adv*

con-de-scen-sion \ˌkän-di-'sen-chən\ *n* [LL *condescension-, condescensio,* fr. *condescensus,* pp. of *condescendere*] (1647) **1** : voluntary descent from one's rank or dignity in relations with an inferior **2** : patronizing attitude or behavior

con-dign \kən-'dīn, 'kän-ˌ\ *adj* [ME *condigne,* fr. MF, fr. L *condignus* very worthy, fr. *com-* + *dignus* worthy — more at DECENT] (15c) : DESERVED, APPROPRIATE ⟨~ punishment⟩ — **con-dign-ly** *adv*

con-di-ment \'kän-də-mənt\ *n* [ME, fr. MF, fr. L *condimentum,* fr. *condire* to pickle, fr. *condere* to build, store up, fr. *com-* + *-dere* to put — more at DO] (15c) : something used to enhance the flavor of food; *esp* : a pungent seasoning — **con-di-men-tal** \ˌkän-də-'ment-ᵊl\ *adj*

con-di-tion \kən-'dish-ən\ *n* [ME *condicion,* fr. MF, fr. L *condicion-, condicio* terms of agreement, condition, fr. *condicere* to agree, fr. *com-* + *dicere* to say, determine — more at DICTION] (14c) **1** **a** : a premise upon which the fulfillment of an agreement depends : STIPULATION **b** *obs* : COVENANT **c** : a provision making the effect of a legal instrument contingent upon an uncertain event; *also* : the event itself **2** : something essential to the appearance or occurrence of something else : PREREQUISITE as **a** : an environmental requirement ⟨available oxygen is an essential ~ for animal life⟩ **b** : the subordinate clause of a conditional sentence **3** **a** : a restricting or modifying factor : QUALIFICATION **b** : an unsatisfactory academic grade that may be raised by

RCL011405A

Case 5:03-cv-04669-JW Document 238-5 Filed 08/27/2004 Page 10 of 11

doing additional work 4 a : a state of being b : social status : RANK c : a usu. defective state of health ⟨a serious heart ~⟩ d : a state of physical fitness or readiness for use ⟨the car was in good ~⟩ ⟨exercising to get into ~⟩ e *pl* : attendant circumstances 5 a *obs* : temper of mind b *obs* : TRAIT c *pl, archaic* : MANNERS, WAYS

²condition *vb* **con-di-tioned; con-di-tion-ing** \-'dish-(ə-)niŋ\ *vt, archaic* (15c) : to make stipulations ~ *vt* 1 : to agree by stipulating 2 : to make conditional 3 a : to put into a proper state for work or use b : AIR-CONDITION 4 : to give a grade of condition to 5 a : to adapt, modify, or mold so as to conform to an environing culture b : to modify so that an act or response previously associated with one stimulus becomes associated with another — **con-di-tion-able** \-(ə-)nə-bəl\ *adj* — **con-di-tion-er** \-(ə-)nər\ *n*

con-di-tion-al \kən-'dish-nəl, -ən-ᵊl\ *adj* (14c) 1 : subject to, implying, or dependent upon a condition ⟨a ~ promise⟩ 2 : expressing, containing, or implying a supposition ⟨the ~ clause *if he speaks*⟩ 3 a : true only for certain values of the variables or symbols involved ⟨~ equations⟩ b : stating the case when one or more random variables are fixed or one or more events are known ⟨~ frequency distribution⟩ 4 a : CONDITIONED 3 ⟨~ reflex⟩ ⟨~ response⟩ b : established by conditioning as the stimulus eliciting a conditional response — **conditional** *n* — **con-di-tion-al-i-ty** \-,dish-ə-'nal-ət-ē\ *n* — **con-di-tion-al-ly** \-'dish-nə-lē, -ən-ᵊl-ē\ *adv*

conditional probability *n* (1961) : the probability that a given event will occur if it is certain that another event has taken place or will take place

con-di-tioned *adj* (1656) 1 : CONDITIONAL 2 : brought or put into a specified state 3 : determined or established by conditioning

con-do \'kän-(,)dō\ *n* [by shortening] (1972) : CONDOMINIUM 3

con-dole \kən-'dōl\ *vb* **con-doled; con-dol-ing** [LL *condolēre*, fr. L *com-* + *dolēre* to feel pain; akin to Gk *daidalos* ingeniously formed] *vi* (1590) 1 *obs* : GRIEVE 2 : to express sympathetic sorrow ⟨we ~ with you in your misfortune⟩ ~ *vt, archaic* : LAMENT, GRIEVE — **con-do-la-to-ry** \-'dō-lə-,tōr-ē, -,tòr-\ *adj*

con-do-lence \kən-'dō-lən(t)s\ *n* (1603) 1 : sympathy with another in sorrow 2 : an expression of sympathy

con-dom \'kän-dəm, 'kən-\ *n* [origin unknown] (1706) : a sheath commonly of rubber worn over the penis (as to prevent conception or venereal infection during coitus)

con-do-min-i-um \,kän-də-'mi-nē-əm\ *n, pl* **-ums** [NL, fr. L *com-* + *dominium* domain] (1705) 1 a : joint dominion; *esp* : joint sovereignty by two or more nations b : a government operating under joint rule 2 : a politically dependent territory under condominium 3 a : individual ownership of a unit in a multiunit structure (as an apartment building) or on land owned in common (as a town house complex); *also* : a unit so owned b : a building containing condominiums

con-do-na-tion \,kän-də-'nā-shən, -dō-\ *n* (1625) : implied pardon of an offense by treating the offender as if it had not been committed

con-done \kən-'dōn\ *vt* **con-doned; con-don-ing** [L *condonare* to forgive, fr. *com-* + *donare* to give — more at DONATION] (1857) : to pardon or overlook voluntarily; *esp* : to treat as if trivial, harmless, or of no importance ⟨~ corruption in politics⟩ *syn* see EXCUSE — **con-don-able** \-'dō-nə-bəl\ *adj* — **con-don-er** *n*

con-dor \'kän-dər, -,dȯ(ə)r\ *n* [Sp *cóndor*, fr. Quechua *kuntur*] (1604) 1 : a very large American vulture (*Vultur gryphus*) of the high Andes having the head and neck bare and the plumage dull black with a downy white neck ruff and white patches on the wings — compare CALIFORNIA CONDOR 2 *pl* **condors** *or* **con-do-res** \kən-'dōr-,ās, -'dōr-\ : a coin (as the centesimo of Chile) bearing the picture of a condor

con-dot-tie-re \,kän-də-'tye(ə)r-ē, ,kän-,dät-ē-'e(ə)r-\ *n, pl* **-tie-ri** \-ē\ [It] (1794) 1 : a leader of a band of mercenaries common in Europe between the 14th and 16th centuries; *also* : a member of such a band 2 : a mercenary soldier

con-duce \kən-'d(y)üs\ *vi* **con-duced; con-duc-ing** [ME *conducen* to conduct, fr. L *conducere* to conduct, conduce, fr. *com-* + *ducere* to lead — more at TOW] (1586) : to lead or tend to a particular and usu. desirable result : CONTRIBUTE

con-du-cive \-'d(y)ü-siv\ *adj* (1646) : tending to promote or assist ⟨an atmosphere ~ to education⟩ — **con-du-cive-ness** *n*

¹con-duct \'kän-(,)dəkt\ *n* [alter. of ME *conduit*, fr. MF, act of leading, escort, fr. ML *conductus*, fr. L *conductus*, pp. of *conducere*] (15c) 1 *obs* : ESCORT, GUIDE 2 : the act, manner, or process of carrying on : MANAGEMENT 3 : a mode or standard of personal behavior esp. as based on moral principles

²con-duct \kən-'dəkt *also* 'kän-,dəkt\ *vt* (15c) 1 : to bring by or as if by leading : GUIDE ⟨~ tourists through a museum⟩ 2 a : to lead from a position of command ⟨~ a siege⟩ ⟨~ a class⟩ b : to direct or take part in the operation or management of ⟨~ an experiment⟩ ⟨~ a business⟩ ⟨~ an investigation⟩ c : to direct the performance of ⟨~ an orchestra⟩ ⟨~ an opera⟩ 3 a : to convey in a channel b : to act as a medium for conveying 4 : to act or behave in a particular and esp. in a controlled or directed manner ~ *vi* 1 *of a road or passage* : to show the way : LEAD 2 a : to act as leader or director b : to have the quality of transmitting light, heat, sound, or electricity — **con-duct-ibil-i-ty** \kən-,dək-tə-'bil-ət-ē\ *n* — **con-duct-ible** \-'dək-tə-bəl\ *adj*

syn CONDUCT, MANAGE, CONTROL, DIRECT mean to use one's powers to lead, guide, or dominate. CONDUCT implies taking responsibility for the acts and achievements of a group; MANAGE implies direct handling and manipulating or maneuvering toward a desired result; CONTROL implies a regulating or restraining in order to keep within bounds or on a course; DIRECT implies constant guiding and regulating so as to achieve smooth operation. *syn* see in addition BEHAVE

con-duc-tance \kən-'dək-tən(t)s\ *n* (1885) 1 : conducting power 2 : the readiness with which a conductor transmits an electric current : the reciprocal of electrical resistance

con-duc-tion \kən-'dək-shən\ *n* (1541) 1 : the act of conducting or conveying 2 : transmission through or by means of a conductor; *also* : CONDUCTIVITY 3 : the transmission of excitation through living tissue and esp. nervous tissue

con-duc-tive \kən-'dək-tiv\ *adj* (1528) : having conductivity : relating to conduction (as of electricity)

con-duc-tiv-i-ty \,kän-,dək-'tiv-ət-ē, kən-\ *n, pl* **-ties** (1837) : the quality or power of conducting or transmitting: as a : the reciprocal of electrical resistivity b : the quality of living matter responsible for the transmission of and progressive reaction to stimuli

con-duc-to-met-ric *or* **con-duc-ti-met-ric** \kən-,dək-tə-'me-trik\ *adj* (ca. 1926) 1 : of or relating to the measurement of conductivity 2 : being or relating to titration based on determination of changes in the electrical conductivity of the solution

con-duc-tor \kən-'dək-tər\ *n* (15c) : one that conducts: as a : GUIDE b : a collector of fares in a public conveyance c : the leader of a musical ensemble d : a substance or body capable of transmitting electricity, heat, or sound — **con-duc-to-ri-al** \,kän-,dək-'tōr-ē-əl, kən-, -'tòr-\ *adj*

con-duc-tress \kən-'dək-trəs\ *n* (1885) : a female conductor

con-duit \'kän-,d(y)ü-ət *also* -d(w)ət\ *n* [ME — more at CONDUCT] (14c) 1 : a natural or artificial channel through which something (as a fluid) is conveyed 2 *archaic* : FOUNTAIN 3 : a pipe, tube, or tile for protecting electric wires or cables 4 : a means of transmitting or distributing ⟨a ~ for illicit payments⟩ ⟨a ~ of information⟩

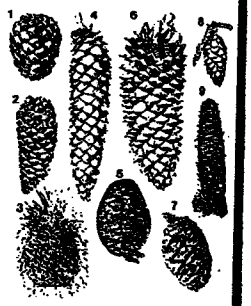
con-du-pli-cate \(ˌ)kän-'d(y)ü-pli-kət\ *adj* [L *conduplicatus*, pp. of *conduplicare* to double, fr. *com-* + *duplic-*, *duplex* double — more at DUPLEX] (1777) : folded lengthwise — used of leaves or petals in the bud — **con-du-pli-ca-tion** \,kän-,d(y)ü-pli-'kā-shən\ *n*

con-dy-lar \'kän-də-lər\ *adj* (1876) : of or relating to a condyle

con-dyle \'kän-,dīl *also* -dᵊl\ *n* [F & L; F, fr. L *condylus* knuckle, fr. Gk *kondylos*] (1634) : an articular prominence of a bone; *esp* : either of a pair that resembles knuckles — **con-dy-loid** \-də-,lȯid\ *adj*

con-dy-lo-ma \,kän-də-'lō-mə\ *n* [NL, fr. Gk *kondylōma*, fr. *kondyl*[os] (ca. 1656) : a warty growth on the skin or adjoining mucous membrane usu. near the anus and genital organs — **con-dy-lo-ma-tous** \-mət-əs\ *adj*

¹cone \'kōn\ *n* [MF or L; MF, fr. L *conus*, fr. Gk *kōnos* — more at HONE] (1562) 1 a : a mass of ovule-bearing or pollen-bearing scales or bracts in trees of the pine family or in cycads that are arranged usu. on a somewhat elongated axis b : any of several flower or fruit clusters suggesting a cone 2 a : a solid generated by rotating a right triangle about one of its legs — called also *right circular cone* b : a solid bounded by a circular or other closed plane base and the surface formed by line segments joining every point of the boundary of the base to a common vertex — see VOLUME table c : a surface traced by a moving straight line passing through a fixed vertex 3 : something that resembles a cone in shape: as a : one of the short sensory end organs of the vertebrate retina that function in color vision b : any of numerous somewhat conical tropical gastropod mollusks (family Conidae) c : the apex of a volcano d : a crisp cone-shaped wafer for holding ice cream



cone 1a: *1* stone pine, *2* cluster pine, *3* Santa Lucia fir, *4* sugar pine, *5* deodar, *6* big-cone spruce, *7* giant sequoia, *8* red spruce, *9* Nordmann's fir

²cone *vt* **coned; con-ing** (ca. 1864) 1 : to make cone-shaped 2 : to bevel like the slanting surface of a cone ⟨~ a tire⟩

cone-flow-er \'kōn-,flau̇(-ə)r\ *n* (1817) : any of several composite plants having cone-shaped flower disks; *esp* : RUDBECKIA

cone-nose \'kōn-,nōz\ *n* (ca. 1891) : any of various large bloodsucking bugs (esp. genus *Triatoma*) including some capable of inflicting painful bites — called also *assassin bug, kissing bug*

con es-pres-sio-ne \,kän-,es-(,)pres-ē-'ō-nē, ,kōn-, -'ō-(,)nā\ *adv* [It, lit., with expression] (ca. 1891) : with feeling — used as a direction in music

Con-es-to-ga \,kän-ə-'stō-gə\ *n* [*Conestoga*, Pa.] (1750) : a broad-wheeled covered wagon drawn usu. by six horses and used esp. for transporting freight across the prairies

co-ney \'kō-nē\ *n, pl* **coneys** [ME *conies*, pl., fr. OF *conis*, pl. of *conil*, fr. L *cuniculus*] (13c) 1 a (1) : RABBIT; *esp* : the European rabbit (*Oryctolagus cuniculus*) (2) : PIKA b : HYRAX c : rabbit fur 2 *obs* : DUPE 3 : any of several fishes; *esp* : a dusky black-spotted reddish-finned grouper (*Cephalopholis fulvus*) of the tropical Atlantic

con-fab \kən-'fab, 'kän-,\ *vi* **con-fabbed; con-fab-bing** (1741) : CONFABULATE — **confab** \'kän-,fab, kən-'\ *n*

con-fab-u-late \kən-'fab-yə-,lāt\ *vi* **-lat-ed; -lat-ing** [L *confabulatus*, pp. of *confabulari*, fr. *com-* + *fabulari* to talk, fr. *fabula* story — more at FABLE] (1613) 1 : CHAT 2 : to hold a discussion : CONFER — **con-fab-u-la-tion** \kən-,fab-yə-'lā-shən, ,kän-\ *n* — **con-fab-u-la-tor** \kən-'fab-yə-,lāt-ər\ *n* — **con-fab-u-la-to-ry** \-lə-,tōr-ē, -,tòr-\ *adj*

con-fect \kən-'fekt\ *vt* [L *confectus*, pp. of *conficere* to prepare — more at COMFIT] (14c) 1 : to put together from varied material ⟨written ~*ing* best sellers⟩ 2 a : PREPARE b : PRESERVE — **con-fect** \'kän-,\ *n*

con-fec-tion \kən-'fek-shən\ *n* (15c) 1 : the act or process of confecting 2 : something confected: as a : a fancy dish or sweetmeat; *also* : a sweet food b : a medicinal preparation usu. made with sugar, syrup, or honey c : a piece of fine craftsmanship

con-fec-tion-ary \-shə-,ner-ē\ *n, pl* **-ar-ies** (1605) 1 *archaic* : CONFECTIONER 2 : CONFECTIONERY 3 3 : SWEETS — **confectionary** *adj*

con-fec-tion-er \-sh(ə-)nər\ *n* (1591) : a manufacturer of or dealer in confections

confectioners' sugar *n* (ca. 1891) : a refined finely powdered sugar

con-fec-tion-ery \-shə-,ner-ē\ *n, pl* **-er-ies** (1545) 1 : sweet foods (as candy or pastry) 2 : the confectioner's art or business 3 : a confectioner's shop

con-fed-er-a-cy \kən-'fed-(ə-)rə-sē\ *n, pl* **-cies** (14c) 1 : a league or compact for mutual support or common action : ALLIANCE 2 : a combination of persons for unlawful purposes : CONSPIRACY 3 : the body formed by persons, states, or nations united by a league; *specif* : the 11 southern states seceding from the U.S. in 1860 and 1861

con-fed-er-al \-(ə-)rəl\ *adj* — **con-fed-er-al-ist** \-ə-st\ *n*

RCL011406

Case 5:03-cv-04669-JW Document 238-5 Filed 08/27/2004 Page 11 of 11

floury \'flau̇-ə-rē, 'flȯ-\ *adj* **floura-i-er; -est** (1839) 1 **a** : of, relating to, or having the characteristics of floss **b** : DOWNY 2 : stylish or glamorous esp. at first impression ⟨slick ~ writing⟩ — **floura-i-ly** \'flau̇-ə-lē\ *adv*

flo-ta \'flōt-ə\ *n* [Sp] (1527) : a fleet of Spanish ships

flo-tage \'flōt-ij\ *n* [*float*] (1626) 1 : FLOTATION 1 2 : material that floats : FLOTSAM 3 *usu* **floatage** : the charge for transferring railroad cars on a barge

flo-ta-tion \flō-'tā-shən\ *n* [*float*] (1806) 1 : the act, process, or state of floating 2 : an act or instance of financing (as an issue of stock) 3 : the separation of the particles of a mass of pulverized ore according to their relative capacity for floating on a given liquid; *also* : any of various similar processes involving the relative capacity of materials for floating 4 : the ability (as of a tire) to stay on the surface of soft ground or snow

flo-til-la \flō-'til-ə\ *n* [Sp, dim. of *flota* fleet, fr. OF *flote*, fr. ON *flotti*; akin to OE *flota* ship, fleet — more at FLOAT] (1711) 1 : a fleet of ships; *esp* : a navy organizational unit consisting of two or more squadrons of small warships 2 : a large force of moving things

flot-sam \'flät-səm\ *n* [AF *floteson*, fr. OF *floter* to float, of Gmc origin; akin to OE *flotian* to float, *flota* ship] (ca. 1607) 1 : floating wreckage of a ship or its cargo; *broadly* : floating debris 2 : a vagrant impoverished people **b** : unimportant miscellaneous material

flounce \'flau̇n(t)s\ *vi* **flounced; flounc-ing** [perh. of Scand origin; akin to Norw *flunsa* to hurry] (1542) 1 **a** : to move with exaggerated jerky motions ⟨little girls *flouncing* about in their mothers' clothes⟩ **b** : to go with sudden determination ⟨*flounced* out of the room in a huff⟩ 2 : FLOUNDER, STRUGGLE

²flounce *n* (1583) : an act or instance of flouncing — **flouncy** \'flau̇n(t)-sē\ *adj*

³flounce *vt* **flounced; flounc-ing** [alter. of earlier *frounce*, fr. ME *frouncen* to curl] (1711) : to trim with flounces

⁴flounce *n* (1713) : a strip of fabric attached by one edge; *also* : a wide ruffle — **flouncy** \'flau̇n(t)-sē\ *adj*

flounc-ing \'flau̇n(t)-siŋ\ *n* (1873) : material used for flounces

flound-der \'flau̇n-dər\ *n, pl* **flounder** *or* **flounders** [ME, of Scand origin; akin to ON *flythra* flounder, *flatr* flat — more at FLAT] (14c) : FLATFISH; *esp* : one of either of two families (Pleuronectidae and Bothidae) that include important marine food fishes

²flounder *vi* **floun-dered; floun-der-ing** \-d(ə-)riŋ\ [prob. alter. of *founder*] (1592) 1 : to struggle to move or obtain footing 2 : to proceed or act clumsily or ineffectually

flour \'flau̇(ə)r\ *n* [ME — more at FLOWER] (13c) 1 : finely ground meal of wheat usu. largely freed from bran; *also* : a similar meal of another material (as a cereal grain, an edible seed, or dried processed fish) 2 : a fine soft powder — **flour-less** *adj* — **floury** \-ə-rē\ *adj*

²flour *vt* (1651) : to coat with or as if with flour ~ *vi* : to break up into particles

flour-ish \'flər-ish, 'flə-rish\ *vb* [ME *florisshen*, fr. MF *floriss-*, stem of *florir*, fr. (assumed) VL *florire*, alter. of L *florēre*, fr. *flor-*, *flos* flower] *vi* (14c) 1 : to grow luxuriantly : THRIVE 2 **a** : to achieve success : PROSPER **b** : to be in a state of activity or production ⟨~*ed* around 1830⟩ **c** : to reach a height of development or influence 3 : to make bold and sweeping gestures ~ *vt* : to wield with dramatic gestures : BRANDISH *syn* see SWING — **flour-ish-er** *n* — **flour-ish-ing-ly** \-iŋ-lē\ *adv*

²flourish *n* (1597) 1 : a period of thriving 2 **a** : an extraneous florid embellishment or passage **b** : an act or instance of brandishing : WAVE **c** : a studied or ostentatious action

flout \'flau̇t\ *vb* [prob. fr. ME *flouten* to play the flute, fr. *floute* flute] *vt* (1551) : to treat with contemptuous disregard : SCORN ⟨~*ing* the rules⟩ ~ *vi* : to indulge in scornful behavior *syn* see SCOFF *usage* see FLAUNT — **flout-er** *n*

²flout *n* (ca. 1570) 1 : INSULT 2 : MOCKERY

¹flow \'flō\ *vb* [ME *flowen*, fr. OE *flōwan*; akin to OHG *flouwen* to rinse, wash, L *pluere* to rain, Gk *plein* to sail, float] *vi* (bef. 12c) 1 **a** (1) : to issue or move in a stream (2) : CIRCULATE **b** : to move with a continual change of place among the constituent particles ⟨the molasses ~*ed* slowly⟩ 2 : RISE ⟨the tide ebbs and ~s⟩ 3 : ABOUND 4 **a** : to proceed smoothly and readily ⟨conversation ~*ed* easily⟩ **b** : to have a smooth continuity ⟨the ~*ing* lines of the car⟩ 5 : to hang loose and billowing 6 : to derive from a source : COME ⟨the wealth that ~s from our industries⟩ 7 : to deform under stress without cracking or rupturing — used esp. of minerals and rocks 8 : MENSTRUATE ~ *vt* 1 **a** : to cause to flow **b** : to cover with water : FLOOD 2 : to discharge in a flow *syn* see SPRING — **flow-ing-ly** \-iŋ-lē\ *adv*

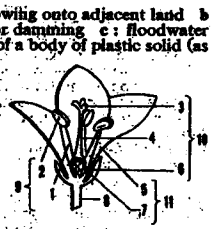
²flow *n* (15c) 1 : an act of flowing 2 : FLOOD 1a, 2 3 **a** : a smooth uninterrupted movement ⟨a constant ~ of information⟩ **b** : STREAM: **a** : a mass of material which has flowed when molten (an old lava ~) **c** : the direction of movement or apparent movement (as of a play in football) 4 : the quantity that flows in a certain time 5 **a** : MENSTRUATION **b** : YIELD, PRODUCTION 6 **a** : the motion characteristic of fluids **b** : a continuous transfer of energy

flow-age \'flō-ij\ *n* (1830) 1 **a** : an overflowing onto adjacent land **b** : a body of water formed by overflowing or damming **c** : floodwater **d** : a stream 2 : gradual deformation of a body of plastic solid (as rock) by intermolecular shear

flow-chart \-,chärt\ *n* (1920) : a diagram that shows step-by-step progression through a procedure or system esp. using connecting lines and a set of conventional symbols — **flow-chart-ing** \-iŋ\ *n*

flow diagram *n* (1943) : FLOWCHART

flow-er \'flau̇(-ə)r\ *n* [ME *flour* flower, best of anything, flour; fr. OF *flor*, *flour*, fr. *flōr-*, *flos* — more at BLOW] (13c) 1 : BLOSSOM, INFLORESCENCE **b** : a shoot of the sporophyte of a higher plant that is specialized for reproduction and consists of a shortened axis bearing modified leaves **c** : a plant cultivated for its blossoms 2 : the best part or example ⟨the ~ of our youth⟩ **b** : the finest most vigorous period **c** : a state of blooming or flourishing 3 *pl* : a finely divided powder produced esp. by condensation or sublimation ⟨~s of sulfur⟩ — **flow-ered** \'flau̇(-ə)rd\ *adj* — **flow-er-ful** \'flau̇(-ə)r-fəl\ *adj* — **flow-er-less** \-ləs\ *adj* — **flow-er-like** \-,līk\ *adj*



cross section of flower 1b: *1* filament, *2* anther, *3* stigma, *4* style, *5* petal, *6* ovary, *7* sepal, *8* pedicel, *9* stamen, *10* pistil, *11* perianth

²flower *vi* (13c) 1 **a** : DEVELOP ⟨~*ed* into young womanhood⟩ **b** : FLOURISH 2 : to produce flowers : BLOSSOM ~ *vt* 1 : to cause to bear flowers 2 : to decorate with floral designs — **flow-er-er** \'flau̇(-ə)r-ər\ *n*

flow-er-age \'flau̇(-ə)r-ij\ *n* (1840) : a flowering state

flower bud *n* (ca. 1847) : a plant bud that produces only a flower

flower bug *n* (ca. 1889) : any of various small mostly black-and-white predaceous bugs (family Anthocoridae) that frequent flowers and feed on pest insects (as aphids and thrips)

flower child *n* [fr. his displaying of flowers as a symbol of his sentiments] (1967) : a hippie who advocates love, beauty, and peace

flow-er-et \'flau̇(-ə)r-ət\ *n* (15c) : FLORET

flower girl *n* (1924) : a little girl who carries flowers at a wedding

flower head *n* (1845) : a capitulum (as of a composite) having sessile flowers so arranged that the whole inflorescence looks like a single flower

flowering dogwood *n* (1843) : a common spring-flowering white-bracted dogwood (*Cornus florida*)

flowering plant *n* (1861) 1 : a plant that produces flowers, fruit, and seed : ANGIOSPERM 2 : a plant notable for or cultivated for its ornamental flowers

flower people *n pl* (1967) : FLOWER CHILDREN

flow-er-pot \'flau̇(-ə)r-,pät\ *n* (1598) : a pot in which to grow plants

flow-ery \'flau̇(-ə)r-ē\ *adj* (14c) 1 : of, relating to, or resembling flowers 2 : marked by or given to rhetorical elegance — **flow-er-i-ness** *n*

¹flown \'flōn\ *past part of* FLY

²flown *adj* [archaic pp. of *¹flow*] (1626) : filled to excess

flow sheet *n* (1912) : FLOWCHART

flow-stone \'flō-,stōn\ *n* (1928) : travertine found where water flowing in a very thin sheet over rocks has deposited mineral matter

flu \'flü\ *n* [by shortening] (1839) : INFLUENZA 2 : any of several virus diseases marked esp. by respiratory symptoms

¹flub \'fləb\ *vb* **flubbed; flub-bing** [origin unknown] *vt* (ca. 1904) : to make a mess of : BOTCH ⟨*flubbed* the test⟩ ~ *vi* : BLUNDER

²flub *n* (1944) : an act or instance of flubbing

flub-dub \'fləb-,dəb\ *n* [origin unknown] (1888) : CLAPTRAP, BUNKUM

fluc-tu-ant \'flək-chə-wənt\ *adj* (1560) 1 : moving in waves 2 : VARIABLE, UNSTABLE 3 : being movable and compressible ⟨a ~ abscess⟩

fluc-tu-ate \'flək-chə-,wāt\ *vb* **-at-ed; -at-ing** [L *fluctuatus*, pp. of *fluctuare*, fr. *fluctus* flow, wave, fr. *fluctus*, pp. of *fluere* — more at FLUID] *vi* (ca. 1656) 1 : to ebb and flow in waves 2 : to shift back and forth uncertainly ~ *vt* : to cause to fluctuate *syn* see SWING — **fluc-tu-a-tion** \,flak-chə-'wā-shən\ *n* — **fluc-tu-a-tion-al** \-'wā-shnəl, -shən-ᵊl\ *adj*

flue \'flü\ *n* [origin unknown] (1582) : an enclosed passageway for directing a current: as **a** : a channel in a chimney for conveying flame and smoke to the outer air **b** : a pipe for conveying flame and hot gases around or through water in a steam boiler **c** : an air channel leading to the lip of a wind instrument **d** : FLUE PIPE

flue-cured \-,kyu̇(ə)rd\ *adj* (1848) : cured with heat transmitted through a flue without exposure to smoke or fumes ⟨~ tobacco⟩

flu-en-cy \'flü-ən-sē\ *n* (1636) : the quality or state of being fluent

flu-ent \'flü-ənt\ *adj* [L *fluient-*, *fluens*, prp. of *fluere*] (1599) 1 **a** : capable of flowing : FLUID **b** : capable of moving with ease and grace ⟨the ~ body of a dancer⟩ 2 **a** : ready or facile in speech ⟨~ in Spanish⟩ **b** : effortlessly smooth and rapid : POLISHED ⟨a ~ performance⟩ — **flu-ent-ly** *adv*

flue pipe *n* (1852) : an organ pipe whose tone is produced by an air current striking the lip and causing the air within to vibrate — compare REED PIPE

flu-er-ic \flü-'er-ik\ *adj* (1967) : FLUIDIC — **flu-er-ics** \-iks\ *n pl but sing in constr*

flue stop *n* (1855) : an organ stop made up of flue pipes

¹fluff \'fləf\ *n* [prob. alter. of *flue* (fluff)] (1790) 1 : NAP, DOWN 2 : something fluffy 3 : something inconsequential 4 : BLUNDER; *esp* : an actor's lapse of memory

²fluff *vt* (1875) 1 : to become fluffy 2 : to make a mistake; *esp* : to forget or bungle one's lines in a play ~ *vt* 1 : to make fluffy 2 **a** : to spoil by a mistake : BOTCH **b** : to deliver badly or forget (one's lines) in a play

fluffy \'fləf-ē\ *adj* **fluff-i-er; -est** (1825) 1 **a** : covered with or resembling fluff **b** : being light and soft or airy ⟨a ~ omelet⟩ 2 : lacking in intellectual content or decisive quality ⟨vague, ~, uncertain policies⟩ — Geoffrey Crowther⟩ — **fluff-i-ly** \'fləf-ə-lē\ *adv* — **fluff-i-ness** \'fləf-ē-nəs\ *n*

flü-gel-horn *or* **flue-gel-horn** \'flü-gəl-,hȯ(ə)rn, 'flü-\ *n* [G, fr. *flügel* wing, flank + *horn* horn; fr. its use to signal the flanking drivers in a battue] (1854) : a valved brass instrument resembling a cornet but having a larger bore — **flü-gel-horn-ist** \-ənst\ *n*

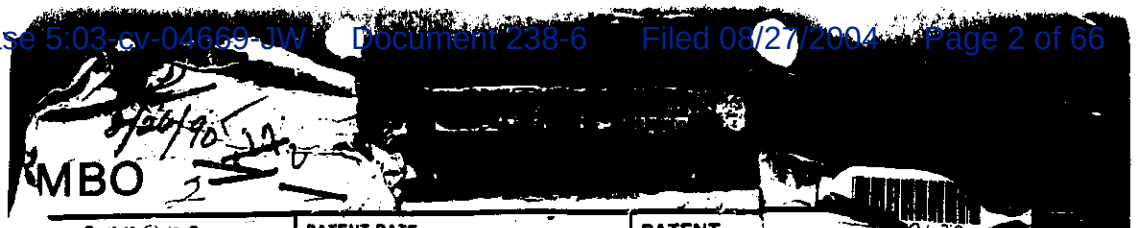
¹flu-id \'flü-əd\ *adj* [F or L; F *fluide*, fr. L *fluidus*, fr. *fluere* to flow; akin to Gk *phlyzein* to boil over] (1603) 1 **a** : having particles that easily move and change their relative position without a separation of the mass and that easily yield to pressure : capable of flowing **b** : likely or tending to change or move ⟨boundaries became ~⟩ 2 : characterized by or employing a smooth easy style ⟨the ballerina's ~ movements⟩ 3 **a** : available for a different use **b** : easily converted into cash ⟨~ assets⟩ — **flu-id-ly** *adv* — **flu-id-ness** *n*

²fluid *n* (1661) : a substance (as a liquid or gas) tending to flow or conform to the outline of its container — **flu-id-al** \-əd-ᵊl\ *adj* — **flu-id-al-ly** \-ᵊl-ē\ *adv*

fluid drive *n* (1941) : an automotive power coupling that operates on a hydraulic turbine principle with the flywheel having a set of turbine blades connected directly to it and driving them in oil thereby turning another set of turbine blades attached to the transmission gears

RCL011407

EXHIBIT 5
(Part 1 of 4)



RMBO

143821	PATENT DATE MAY 1 1990	PATENT NUMBER

SERIAL NUMBER	FILING DATE	CLASS	SUBCLASS	GROUP ART UNIT	EXAMINER
			490	234	TRANS

APPLICANTS

CONTINUING DATA********

None V

FOREIGN/PCT APPLICATIONS********

None V

FOREIGN FILING LICENSE GRANTED 02/09/88

Foreign priority claimed 35 USC 119 conditions met	Verified and Acknowledged	AS FILED	STATE OR COUNTRY	SHEETS DRWGS.	TOTAL CLAIMS	INDEP. CLAIMS	FILING FEE RECEIVED	ATTORNEY'S DOCKET NO.
☐ yes ☒ no / ☐ yes ☒ no	Examiner's Initials	→	CA	15	20	4	$ 542.00	3868-2

ADDRESS

[redacted]
[redacted], CA [redacted]

TITLE

[redacted] INTEGRATED CIRCUITS

U.S. DEPT. of COMM.-Pat. & TM Office — PTO-436L (rev. 10-78)

CERTIFICATE

JAN 14 1991

PARTS OF APPLICATION FILED SEPARATELY

NOTICE OF ALLOWANCE MAILED	PREPARED FOR ISSUE 12-7-89	CLAIMS ALLOWED
11-29-89	V. N. TRANS, Assistant Examiner; Peggy Harrod, Docket Clerk	Total Claims 20; Print Claim

ISSUE FEE			DRAWING		
Amount Due	Date Paid	FELIX D. GRUBER PRIMARY EXAMINER ART UNIT 234 Primary Examiner	Sheets Drwg. 12	Figs. Drwg. 17	Print Fig. 3
620.00	2/21/90				

ISSUE CLASSIFICATION		ISSUE BATCH NUMBER
Class 364	Subclass 490	RCL000001

Label Area

WARNING: The information disclosed herein may be restricted. Unauthorized disclosure may be prohibited by the United States Code Title 35, Sections 122, 181 and 368. Possession outside the U.S. Patent & Trademark Office is restricted to authorized employees and contractors only.

APPROVED FOR LICEN

INITIALS _____

14[illegible]21

[redacted]

CONTENTS

Entered or Counted		Received or Mailed
	1. Application ✓ papers.	
	2. Prior Art	Jan. 13, 1988
	3. ~~Letter~~ Rejection (35 USC 3mos.)	JAN 18, 1989
	4. Power To Inspect	Jan. 31, 1989
	5. Prior Art	April 20, 198[illegible]
	6. Amdt. A	April 20, 198[illegible]
	7. Final Rej. (3mos)	Aug. 15, 198[illegible]
	8. Letter Re Interview	Oct. 9, 1989
	9. Amdt. B (E)	Nov. 20, 198[illegible]
	10. Notice of Allow	Nov 29, 1989
	11. Issue Fee (12)	1-4-90
	12. PTO GRANT MAY 01 1990	
	13. Rep Corr. C.R. 322	9-18-90
	14.	
	15.	
	16.	
	17.	
	18.	
	19.	
	20.	
	21.	
	22.	
	23.	
	24.	
	25.	
	26.	
	27.	
	28.	
	29.	
	30.	
	31.	
	32.	

RCL000002

ORIGINAL CLASSIFICATION

CLASS	SUBCLASS
364	490

TION SERIAL NUMBER
143,821

NT'S NAME (PLEASE PRINT)
OBAYASHI et al.

CROSS REFERENCE(S)

CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)			
364	489	488	521	

CF 15/60

ART UNIT 234

TYPE PLEASE STAMP OR PRINT FULL NAME.
Y N TRANS

PRIMARY EXAMINER (PLEASE STAMP OR PRINT FULL NAME)
ALEX D. GRUBER
PRIMARY EXAMINER

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE

ISSUE CLASSIFICATION SLIP

INDEX OF CLAIMS

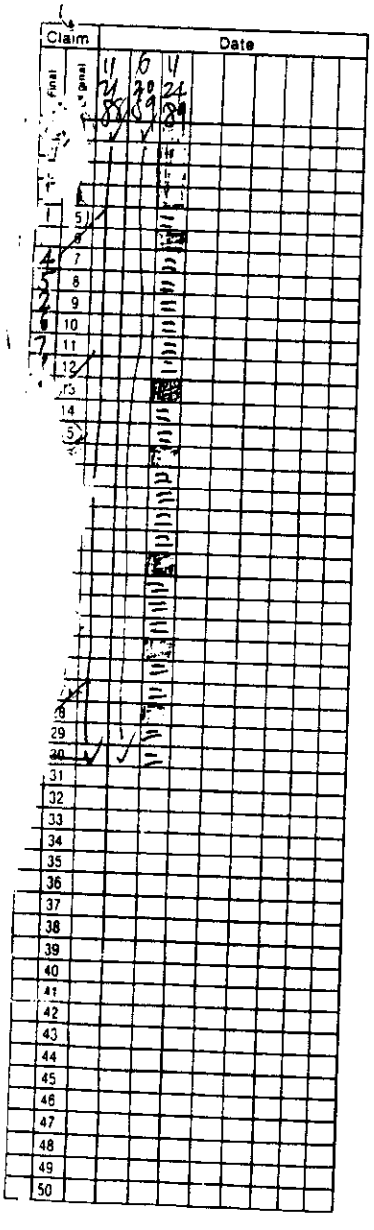
Claim	Date									
Final	Original									
	1	4/24/89								
	2	6/29/89								
	3	4/24/89								
	4									
	5									
	6									
	7									
	8									
	9									
	10									
	11									
	12									
	13									
	14									
	15									
	16									
	17									
	18									
	19									
	20									
	21									
	22									
	23									
	24									
	25									
	26									
	27									
	28									
	29									
	30									
	31									
	32									
	33									
	34									
	35									
	36									
	37									
	38									
	39									
	40									
	41									
	42									
	43									
	44									
	45									
	46									
	47									
	48									
	49									
	50									

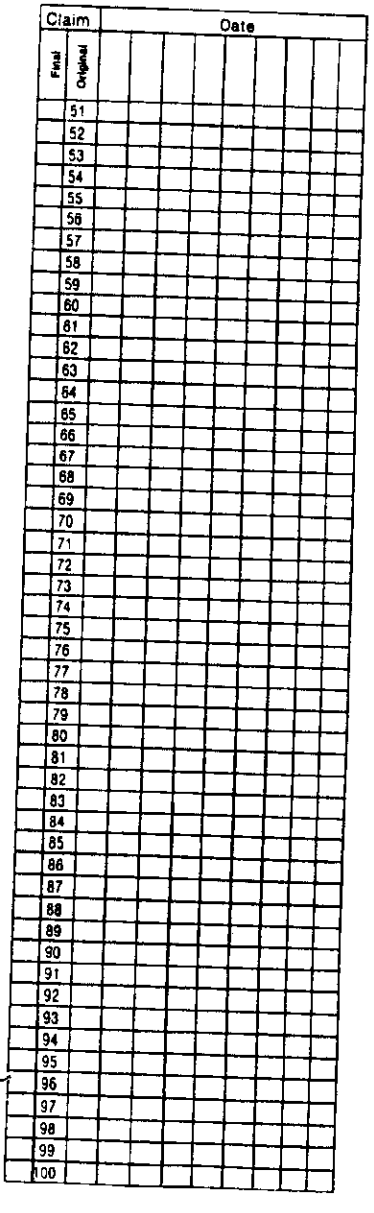
Claim	Date									
Final	Original									
	51									
	52									
	53									
	54									
	55									
	56									
	57									
	58									
	59									
	60									
	61									
	62									
	63									
	64									
	65									
	66									
	67									
	68									
	69									
	70									
	71									
	72									
	73									
	74									
	75									
	76									
	77									
	78									
	79									
	80									
	81									
	82									
	83									
	84									
	85									
	86									
	87									
	88									
	89									
	90									
	91									
	92									
	93									
	94									
	95									
	96									
	97									
	98									
	99									
	100									

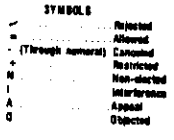
SYMBOLS

=	Rejected
a	Allowed
- (Through numeral)	Cancelled
+	Restricted
N	Non-elected
I	Interference
A	Appeal
O	Objected

INDEX OF CLAIMS







RCL000004

[redacted]

SEARCHED

Class	Sub.	Date	Exmr.
364	488-491	11/18/88	CJ
	521		
	300		
Updated		6/29/89	CJ
364	513	11/24/89	CJ
updated		11/4/99	CJ

SEARCH NOTES

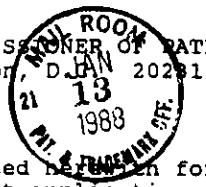
	Date	Exmr.

INTERFERENCE SEARCHED

Class	Sub.	Date	Exmr.
364	488-491	11/4/88	CJ
	521		
	300		

RCL000005

CA. DOCKET NO.: 3868-2
ORDER NO.: 5749
DATE: January 13, 1988



THE COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

"Express Mail" mailing label number B47691822
Date of Deposit January 13, 1988

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D. C. 20231.

James L. Hicks

Sir:

Transmitted herewith for filing is the patent application of:

Inventor(s): Hideaki Kobayashi and
Masahiro Shindo

For: KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS

Enclosed are:
[X] 3 sets of formal drawings (15 sheets to a set)
[X] An assignment of the invention to: International Chip Corporation and Ricoh Company, Ltd.
[ ] A verified statement to establish small entity status under 37 CFR 1.9 and 37 CFR 1.27
[X] Information Disclosure Statement
The filing fee has been calculated as shown below:

FOR:	(Col. 1) NO. FILED	(Col. 2) NO. EXTRA	SMALL ENTITY RATE	SMALL ENTITY FEE		OTHER THAN SMALL ENTITY RATE	OTHER THAN SMALL ENTITY FEE
BASIC FEE:				$ 170			$ 340
TOTAL CLAIMS:	30 - 20 = *	10	x 6 =	$	OR	x 12 =	$ 120
INDEP CLAIMS:	6 - 3 = *	3	x 17 =	$	OR	x 34 =	$ 102
[ ] MULTIPLE DEPENDENT CLAIM PRESENTED			+ 55 =	$	OR	+110 =	$ 0
*If the difference in Col. 1 is less than zero, enter "0" in Col. 2.			TOTAL:	$	OR	TOTAL:	$ 562

[ ] A check in the amount of $_____ to cover the filing fee is enclosed.

[X] A check in the amount of $ 569.00 to cover the filing fee PLUS THE ASSIGNMENT RECORDING FEE (additional $7) is enclosed.

[X] The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 16-0605. A duplicate copy of this sheet is enclosed.
[X] Any additional filing fees required under 37 CFR 1.16.
[X] Any patent application processing fees under 37 CFR 1.17.

[X] The Commissioner is hereby authorized to charge payment of the following fees during the pendency of this application or credit any overpayment to Deposit Account No. 16-0605. A duplicate of this sheet is enclosed.
[X] Any patent application processing fees under 37 CFR 1.17.
[ ] The issue fee set in 37 CFR 1.18 at or before mailing of the Notice of Allowance, pursuant to 37 CFR 1.311(b).
[X] Any filing fees under 37 CFR 1.16 for presentation of extra claims.

Respectfully submitted,

Raymond O. Linker, Jr. Reg. No. 26,419
Attorney of Record

Bell, Seltzer, Park & Gibson, PA
Post Office Drawer 34009
Charlotte, North Carolina 28234
Telephone: (704) 377-1561

RCL000006

PATENT APPLICATION SERIAL NO. 143821

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

080	01/27/88	143821		
080	01/27/88	143821	1 101	340.00 CK
080	01/27/88	143821	1 103	120.00 CK
			1 102	102.00 CK

PTO-1556
(5/87)

RCL000007

MAIL ROOM
JAN
13
1988

KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS

Field and Background of the Invention

This invention relates to the design of integrated circuits, and more particularly relates to a computer-aided method and apparatus for designing integrated circuits.

An application specific integrated circuit (ASIC) is an integrated circuit chip designed to perform a specific function, as distinguished from standard, general purpose integrated circuit chips, such as microprocessors, memory chips, etc. A highly skilled design engineer having specialized knowledge in VLSI circuit design is ordinarily required to design an ASIC. In the design process, the VLSI design engineer will consider the particular objectives to be accomplished and tasks to be performed by the integrated circuit and will create structural level design specifications which define the various hardware components required to perform the desired function, as well as the interconnection requirements between these components. A system controller must also be designed for synchronizing the operations of these components. This requires an extensive and all encompassing knowledge of the various hardware components required to achieve the desired objectives, as well as their interconnection requirements, signal level compatibility, timing compatibility, physical layout, etc. At each design step, the designer must do tedious analysis. The design specifications created by the VLSI design engineer may, for example, be in the form of circuit schematics, parameters or specialized hardware description languages (HDLs).

From the structural level design specifications, the description of the hardware components and interconnections is

converted to a physical chip layout level description which describes the actual topological characteristics of the integrated circuit chip. This physical chip layout level description provides the mask data needed for fabricating the chip.

Due to the tremendous advances in very large scale integration (VLSI) technology, highly complex circuit systems are being built on a single chip. With their complexity and the demand to design custom chips at a faster rate, in large quantities, and for an ever increasing number of specific applications, computer-aided design (CAD) techniques need to be used. CAD techniques have been used with success in design and verification of integrated circuits, at both the structural level and at the physical layout level. For example, CAD systems have been developed for assisting in converting VLSI structural level descriptions of integrated circuits into the physical layout level topological mask data required for actually producing the chip. Although the presently available computer-aided design systems greatly facilitate the design process, the current practice still requires highly skilled VLSI design engineers to create the necessary structural level hardware descriptions.

There is only a small number of VLSI designers who possess the highly specialized skills needed to create structural level integrated circuit hardware descriptions. Even with the assistance of available VLSI CAD tools, the design process is time consuming and the probability of error is also high because of human involvements. There is a very significant need for a better and more cost effective way to design custom integrated circuits.

Summary of the Invention

In accordance with the present invention a CAD (computer-aided design) system and method is provided which enables a user to define the functional requirements for a desired target integrated circuit, using an easily understood functional level representation, and which generates therefrom the detailed information needed for directly producing an application specific integrated circuit (ASIC) to carry out those specific functions. Thus, the present invention, for the first time, opens the possibility for the design and production of ASICs by designers, engineers and technicians who may not possess the specialized expert knowledge of a highly skilled VLSI design engineer.

The functional specifications of the desired ASIC can be defined in a suitable manner, such as in list form or preferably in a flowchart format. The flowchart is a highly effective means of describing a sequence of logical operations, and is well understood by software and hardware designers of varying levels of expertise and training. From the flowchart (or other functional specifications), the system and method of the present invention translates the functional specifications into a structural level definition of an integrated circuit, which can be used directly to produce the ASIC. The structural level definition includes a list of the integrated circuit hardware cells needed to achieve the functional specifications. These cells are selected from a cell library of previously designed hardware cells of various functions and technical specifications. The system also generates data paths among the selected hardware cells. In addition, the present invention generates a system controller and control paths for the selected integrated circuit

hardware cells. The list of hardware cells and their interconnection requirements may be represented in the form of a netlist. From the netlist it is possible using either known manual techniques or existing VLSI CAD layout systems to generate the detailed chip level geometrical information (e.g. mask data) required to produce the particular application specific integrated circuit in chip form.

The preferred embodiment of the system and method of the present invention which is described more fully hereinafter is referred to as a Knowledge Based Silicon Compiler (KBSC). The KBSC is an ASIC design methodology based upon artificial intelligence and expert systems technology. The user interface of KBSC is a flowchart editor which allows the designer to represent VLSI systems in the form of a flowchart. The KBSC utilizes a knowledge based expert system, with a knowledge base extracted from expert ASIC designers with a high level of expertise in VLSI design to generate from the flowchart a netlist which describes the selected hardware cells and their interconnection requirements.

Brief Description of the Drawings

The invention will be better understood by reference to the detailed description which follows, taken in connection with the accompanying drawings, in which --

Figure 1a illustrates a functional level design representation of a portion of a desired target circuit, shown in the form of a flowchart;

Figure 1b illustrates a structural level design representation of an integrated circuit;

Figure 1c illustrates a design representation of a circuit at a physical layout level, such as would be utilized in the fabrication of an integrated circuit chip;

Figure 2 is a block schematic diagram showing how integrated circuit mask data is created from flowchart descriptions by the KBSC system of the present invention;

Figure 3 is a somewhat more detailed schematic illustration showing the primary components of the KBSC system;

Figure 4 is a schematic illustration showing how the ASIC design system of the present invention draws upon selected predefined integrated circuit hardware cells from a cell library;

Figure 5 is an example flowchart defining a sequence of functional operations to be performed by an integrated circuit;

Figure 6 is a structural representation showing the hardware blocks and interconnection requirements for the integrated circuit defined in Figure 5;

Figure 7 is an illustration of the flowchart editor window;

Figure 8 is an illustration of the flowchart simulator window;

Figure 9 is an illustration of the steps involved in cell list generation;

Figure 10 is an example flowchart for a vending machine system;

Figure 11 illustrates the hardware components which correspond to each of the three macros used in the flowchart of Figure 10;

Figure 12 is an initial block diagram showing the hardware components for an integrated circuit as defined in the flowchart of Figure 10;

Figure 13 is a block diagram corresponding to Figure 12 showing the interconnections between blocks;

Figure 14 is a block diagram corresponding to Figure 13 after register optimization; and

Figure 15 is a block diagram corresponding to Figure 14 after further optimization.

Detailed Description of Illustrative Embodiments

Figures 1a, 1b and 1c illustrate three different levels of representing the design of an integrated circuit. Figure 1a shows a functional (or behavioral) representation in the form of a flowchart. A flowchart is a graphic representation of an algorithm and consists of two kinds of blocks or states, namely actions and conditions (decisions). Actions are conventionally represented in the flowchart by a rectangle, and conditions are represented by a diamond. Transitions between actions and conditions are represented by lines with arrows. Figure 1b illustrates a structural (or logic) level representation of an integrated circuit. In this representation, blocks are used to represent integrated circuit hardware components for performing various functions, and the lines interconnecting the blocks represent paths for the flow of data or control signals between the blocks. The blocks may, for example, represent hardware components such as adders, comparators, registers, system controllers, etc. Figure 1c illustrates a physical layout level representation of an integrated circuit design, which provides the detailed mask data necessary to actually manufacture the devices and conductors which together comprise integrated circuit.

As noted earlier, the design of an integrated circuit at the structural level requires a design engineer with highly

specialized skills and expertise in VLSI design. In the KBSC system of the present invention, however, integrated circuits can be designed at a functional level because the expertise in VLSI design is provided and applied by the invention. Allowing the designer to work with flowcharts instead of logic circuit schematics simplifies the task of designing custom integrated circuits, making it quicker, less expensive and more reliable. The designer deals with an algorithm using simple flowcharts at a behavioral level, and needs to know only the necessary logical steps to complete a task, rather than the specific means for accomplishing the task. Designing with flowcharts requires less work in testing because flowcharts allow the designer to work much closer to the algorithm. On the other hand, previously existing VLSI design tools require the designer to represent an algorithm with complex circuit schematics at a structural level, therefore requiring more work in testing. Circuit schematics make it harder for the designer to cope with the algorithm function which needs to be incorporated into the target design because they intermix the hardware and functional considerations. Using flowcharts to design custom integrated circuits will allow a large number of system designers to access VLSI technology, where previously only a small number of designers had the knowledge and skills to create the necessary structural level hardware descriptions.

The overall system flow is illustrated in Figure 2. The user enters the functional specifications of the circuit into the knowledge based silicon compiler (KBSC) 10 in the form of a flowchart 11. The KBSC 10 then generates a netlist 15 from the flowchart. The netlist 15 includes a custom generated system controller, all other hardware cells required to implement the

necessary operations, and interconnection information for connecting the hardware cells and the system controller. The netlist can be used as input to any existing VLSI layout and routing tool 16 to create mask data 18 for geometrical layout.

System Overview

The primary elements or modules which comprise the KBSC system are shown in Figure 3. In the embodiment illustrated and described herein, these elements or modules are in the form of software programs, although persons skilled in the appropriate art will recognize that these elements can easily be embodied in other forms, such as in hardware.

Referring more particularly to Figure 3, it will be seen that the KBSC system 10 includes a program 20 called EDSIM, which comprises a flowchart editor 21 for creating and editing flowcharts and a flowchart simulator 22 for simulation and verification of flowcharts. Actions to be performed by each of the rectangles represented in the flowchart are selected from a macro library 23. A program 30 called PSCS (path synthesizer and cell selector) includes a data and control path synthesizer module 31, which is a knowledge based system for data and control path synthesis. PSCS also includes a cell selector 32 for selecting the cells required for system design. The cell selector 32 selects from a cell library 34 of previously designed hardware cells the appropriate cell or cells required to perform each action and condition represented in the flowchart. A controller generator 33 generates a custom designed system controller for controlling the operations of the other hardware cells. The knowledge base 35 contains ASIC design expert knowledge required for data path synthesis and cell selection. Thus, with a functional flowchart input, PSCS generates a system

controller, selects all other hardware cells, generates data and control paths, and generates a netlist describing all of this design information.

The KBSC system employs a hierarchal cell selection ASIC design approach, as is illustrated in Figure 4. Rather than generating every required hardware cell from scratch, the system draws upon a cell library 34 of previously designed, tested and proven hardware cells of various types and of various functional capabilities with a given type. The macro library 23 contains a set of macros defining various actions which can be specified in the flowchart. For each macro function in the macro library 23 there may be several hardware cells in the cell library 34 of differing geometry and characteristics capable of performing the specified function. Using a rule based expert system with a knowledge base 35 extracted from expert ASIC designers, the KBSC system selects from the cell library 34 the optimum cell for carrying out the desired function.

Referring again to Figure 3, the cells selected by the cell selector 32, the controller information generated by the controller generator 33 and the data and control paths generated by the data/control path synthesizer 31 are all utilized by the PSCS program 30 to generate the netlist 15. The netlist is a list which identifies each block in the circuit and the interconnections between the respective inputs and outputs of each block. The netlist provides all the necessary information required to produce the integrated circuit. Computer-aided design systems for cell placement and routing are commercially available which will receive netlist data as input and will lay out the respective cells in the chip, generate the necessary

routing, and produce mask data which can be directly used by a chip foundry in the fabrication of integrated circuits.

System Requirements

The KBSC system can be operated on a suitable programed general purpose digital computer. By way of example, one embodiment of the system is operated in a work station environment such as Sun3 and VAXStation-II/GPX Running UNIX Operating System and X Window Manager. The work station requires a minimum of 8 megabytes of main storage and 20 megabytes of hard disk space. The monitor used is a color screen with 8-bit planes. The software uses C programming language and INGRES relational data base.

The human interface is mainly done by the use of pop up menus, buttons, and a special purpose command language. The permanent data of the integrated circuit design are stored in the data base for easy retrieval and update. Main memory stores the next data temporarily, executable code, design data (flowchart, logic, etc.), data base (cell library), and knowledge base. The CPU performs the main tasks of creating and simulating flowcharts and the automatic synthesis of the design.

Flowchart Example

To describe the mapping of a flowchart to a netlist, consider an example flowchart shown in Figure 5, which is of part of a larger overall system. In this illustrative flowchart, two variables, VAL1 and VAL2 are compared and if they are equal, they are added together. In this instance, the first action (Action 1) involves moving the value of variable VAL1 to register A. The second action comprises moving the value of variable VAL2 to register B. Condition 1 comprises comparing the values in

registers A and B. Action 3 comprises adding the values of registers A and B and storing the result in register C.

In producing an integrated circuit to carry out the function defined in Figure 5, the KBSC maps the flowchart description of the behavior of the system to interconnection requirements between hardware cells. The hardware cells are controlled by a system controller which generates all control signals. There are two types of variables involved in a system controller:

(1) Input variables: These are generated by hardware cells, and/or are external input to the controller. These correspond to conditions in the flowchart.

(2) Output variables: These are generated by the system controller and correspond to actions in the flowchart.

Figure 6 illustrates the results of mapping the flowchart of Figure 5 onto hardware cells. The actions and the conditions in the flowchart are used for cell selection and data and control path synthesis. The VAL1 register and VAL2 register and the data paths leading therefrom have already been allocated in actions occurring before Action 1 in our example. Action 1 causes generation of the data register A. Similarly, Action 2 causes the allocation of data register B. The comparator is allocated as a result of the comparison operation in Condition 1. The comparison operation is accomplished by 1) selecting a comparator cell, 2) mapping the inputs of the comparator cell to registers A and B, 3) generating data paths to connect the comparator with the registers A and B and 4) generating input variables corresponding to equal to, greater than, and less than for the system controller. Similarly the add operation in Action

3 causes selection of the adder cell, mapping of the adder parameters to the registers and creating the data paths.

Following this methodology, a block list can be generated for a given flowchart. This block list consists of a system controller and as many other blocks as may be required for performing the necessary operations. The blocks are connected with data paths, and the blocks are controlled by the system controller through control paths. These blocks can be mapped to the cells selected from a cell library to produce a cell list.

Interactive Flowchart Editor and Simulator

The creation and verification of the flowchart is the first step in the VLSI design methodology. The translation from an algorithm to an equivalent flowchart is performed with the Flowchart Editor 21 (Figure 3). The verification of the edited flowchart is performed by the Flowchart Simulator 22. The Flowchart Editor and Simulator are integrated into one working environment for interactive flowchart editing, with a designer friendly interface.

EDSIM is the program which contains the Flowchart Editor 21 and the Flowchart Simulator 22. It also provides functions such as loading and saving flowcharts. EDSIM will generate an intermediate file, called a statelist, for each flowchart. This file is then used by the PSCS program 30 to generate a netlist.

Flowchart Editor

The Flowchart Editor 21 is a software module used for displaying, creating, and editing the flowchart. This module is controlled through the flowchart editing window illustrated in Figure 7. Along with editing functions the Flowchart Editor also provides checking of design errors.

The following is a description of the operations of the Flowchart Editor. The main editing functions include, create, edit, and delete states, conditions, and transitions. The create operation allows the designer to add a new state, condition, or transitions to a flowchart. Edit allows the designer to change the position of a state, condition or transition, and delete allows the designer to remove a state, condition or transition from the current flowchart. States which contain actions are represented by boxes, conditions are represented by diamonds, and transitions are represented by lines with arrows showing the direction of the transition.

Edit actions allows the designer to assign actions to each box. These actions are made up of macro names and arguments. An example of arguments is the setting and clearing of external signals. A list of basic macros available in the macro library 23 is shown in Table 1.

TABLE 1

Macro	Description
ADD (A,B,C)	C = A + B
SUB (A,B,C)	C = A - B
MULT (A,B,C)	C = A * B
DIV (A,B,C)	C = A div B
DECR (A)	A = A - 1
INCR (A)	A = A + 1
CLR (A)	A = 0
REG (A,B)	B = A
CMP (A,B)	Compare A to B and set EQ,LT,GT signals
CMP0 (A)	Compare A to 0 and set EQ,LT,GT signals
NEGATE (A)	A = NOT (A)
MOD (A,B,C)	C = A Modulus B
POW (A,B,C)	C = A^B
DC2 (A,S1,S2,S3,S4)	Decode A into S1,S2,S3,S4
EC2 (S1,S2,S3,S4,A)	Encode S1,S2,S3,S4 into A
MOVE (A,B)	B = A
CALL sub-flowchart (A,B,...)	Call a sub-flowchart. Pass A,B...
START (A,B,...)	Beginning state of a sub-flowchart
STOP (A,B...)	Ending state of a sub-flowchart

The Flowchart Editor also provides a graphical display of the flowchart as the Flowchart Simulator simulates the flowchart. This graphical display consists of boxes, diamonds, and lines as shown in Figure 7. All are drawn on the screen and look like a traditional flowchart. By displaying the flowchart

on the screen during simulation it allows the designer to design and verify the flowchart at the same time.

Flowchart Simulator

The Flowchart Simulator 22 is a software module used for simulating flowcharts. This module is controlled through the simulator window illustrated in Figure 8. The Flowchart Simulator simulates the transitions between states and conditions in a flowchart. The following is a list of the operations of the Flowchart Simulator:

edit data - Change the value of a register or memory.

set state - Set the next state to be simulated.

set detail or summary display - Display summary or detail information during simulation.

set breaks - Set a breakpoint.

clear breaks - Clear all breakpoints.

show breaks - Display current breakpoints.

step - Step through one transition.

execute - Execute the flowchart.

stop - Stop executing of the flowchart.

history ON or history OFF - Set history recording on or off.

cancel - Cancel current operation.

help - Display help screen.

close - Close the simulator window.

The results of the simulation are displayed within the simulator window. Also the editor window will track the flowchart as it is being simulated. This tracking of the flowchart makes it easy to edit the flowchart when an error is found.

Cell Selection

The Cell Selector 32 is a knowledge based system for selecting a set of optimum cells from the cell library 34 to implement a VLSI system. The selection is based on functional descriptions in the flowchart, as specified by the macros assigned to each action represented in the flowchart. The cells selected for implementing a VLSI system depend on factors such as cell function, fabrication technology used, power limitations, time delays etc. The cell selector uses a knowledge base extracted from VLSI design experts to make the cell selection.

To design a VLSI system from a flowchart description of a user application, it is necessary to match the functions in a flowchart with cells from a cell library. This mapping needs the use of artificial intelligence techniques because the cell selection process is complicated and is done on the basis of a number of design parameters and constraints. The concept used for cell selection is analogous to that used in software compilation. In software compilation a number of subroutines are linked from libraries. In the design of VLSI systems, a functional macro can be mapped to library cell.

Figure 4 illustrates the concept of hierarchical cell selection. The cell selection process is performed in two steps:

(1) selection of functional macros

(2) selection of geometrical cells

A set of basic macros is shown in Table 1. A macro corresponds to an action in the flowchart. As an example, consider the operation of adding A and B and storing the result in C. This function is mapped to the addition macro ADD(X, Y, Z). The flowchart editor and flowchart simulator are used to draw the rectangles, diamonds and lines of the flowchart, to

assign a macro selected from the macro library 23 to each action represented in the flowchart, and to verify the functions in flowcharts. The flowchart is converted into an intermediate form (statelist) and input to the Cell Selector.

The Cell Selector uses a rule based expert system to select the appropriate cell or cells to perform each action. If the cell library has a number of cells with different geometries for performing the operation specified by the macro, then an appropriate cell can be selected on the basis of factors such as cell function, process technology used, time delay, power consumption, etc.

The knowledge base of Cell Selector 32 contains information (rules) relating to:

(1) selection of macros

(2) merging two macros

(3) mapping of macros to cells

(4) merging two cells

(5) error diagnostics

The above information is stored in the knowledge base 35 as rules.

Cell List Generation

Figure 9 shows the cell list generation steps. The first step of cell list generation is the transformation of the flowchart description into a structure that can be used by the Cell Selector. This structure is called the statelist. The blocklist is generated from the statelist by the inference engine. The blocklist contains a list of the functional blocks to be used in the integrated circuit. Rules of the following type are applied during this stage.

- map arguments to data paths
- map actions to macros
- connect these blocks

Rules also provide for optimization and error diagnostics at this level.

The cell selector maps the blocks to cells selected from the cell library 34. It selects an optimum cell for a block. This involves the formulation of rules for selecting appropriate cells from the cell library. Four types of information are stored for each cell. These are:

(1) functional level information: description of the cell at the register transfer level.

(2) logic level information: description in terms of flip-flops and gates.

(3) circuit level information: description at the transistor level.

(4) Layout level information: geometrical mask level specification.

The attributes of a cell are:

- cell name
- description
- function
- width
- height
- status
- technology
- minimum delay
- typical delay
- maximum delay
- power

- file
- designer
- date
- comment
- inspector

In the cell selection process, the above information can be used. Some parameters that can be used to map macros to cells are:

(1) name of macro
(2) function to be performed
(3) complexity of the chip
(4) fabrication technology
(5) delay time allowed
(6) power consumption
(7) bit size of macro data paths

Netlist Generation

The netlist is generated after the cells have been selected by PSCS. PSCS also uses the macro definitions for connecting the cell terminals to other cells. PSCS uses the state-to-state transition information from an intermediate form representation of a flowchart (i.e. the statelist) to generate a netlist. PSCS contains the following knowledge for netlist generation:

1) Data path synthesis
2) Data path optimization
3) Macro definitions
4) Cell library
5) Error detection and correction

The above information is stored in the knowledge base 35 as rules. Knowledge engineers help in the formulation of

these rules from ASIC design experts. The macro library 23 and the cell library 34 are stored in a database of KBSC.

A number of operations are performed by PSCS. The following is a top level description of PSCS operations:

(1) Read the flowchart intermediate file and build a statelist.

(2) current_context = START

(3) Start the inference engine and load the current context rules.

(4) Perform one of the following operations depending upon current_context:

(a) Modify the statelist for correct implementation.

(b) Create blocklist, macrolist and datapaths.

(c) Optimize blocklist and datapath list and perform error checks.

(d) Convert blocks to cells.

(e) Optimize cell list and perform error checks.

(f) Generate netlist.

(g) Optimize netlist and perform error checks and upon completion Goto 7.

(5) If current_context has changed, load new context rules.

(6) Goto 4.

(7) Output netlist file and stf files and Stop.

In the following sections, operations mentioned in step 4 are described. The Rule Language and PSCS display are also described.

Rule Language

The rule language of PSCS is designed to be declarative and to facilitate rule editing. In order to make the expert understand the structure of the knowledge base, the rule language provides means for knowledge representation. This will enable the format of data structures to be stated in the rule base, which will enable the expert to refer to them and understand the various structures used by the system. For example, the expert can analyze the structure of wire and determine its components. The expert can then refer these components into rules. If a new object has to be defined, then the expert can declare a new structure and modify some existing structure to link to this new structure. In this way, the growth of the data structures can be visualized better by the expert. This in turn helps the designer to update and append rules.

The following features are included in the rule language:

i) Knowledge representation in the form of a record structure.

ii) Conditional expressions in the antecedent of a rule.

iii) Facility to create and destroy structure in rule actions.

iv) The assignment statement in the action of a rule.

v) Facility for input and output in rule actions.

vi) Provide facility to invoke C functions from rule actions.

The rule format to be used is as follows:

```
Rule <number> <context>
If {
     <if-clause>
   }
Then {
     <then-clause>
   }

where <number>      : rule number
      <context>     : context in which this rule is
                      active
      <if-clause>   : the condition part of the rule
      <then-clause> : the action part of the rule
```

Inference Strategy

The inference strategy is based on a fast pattern matching algorithm. The rules are stored in a network and the requirement to iterate through the rules is avoided. This speeds up the execution. The conflict resolution strategy to be used is based on the following:

(1) The rule containing the most recent data is selected.

(2) The rule which has the most complex condition is selected.

(3) The rule declared first is selected.

Rule Editor

PSCS provides an interactive rule editor which enables the expert to update the rule set. The rules are stored in a database so that editing capabilities of the database package can be used for rule editing. To perform this operation the expert needs to be familiar with the various knowledge structures and the inferencing process. If this is not possible, then the help of a knowledge engineer is needed.

PSCS provides a menu from which various options can be set. Mechanisms are provided for setting various debugging flags and display options, and for the overall control of PSCS.

Facility is provided to save and display the blocklist created by the user. The blocklist configuration created by the user can be saved in a file and later be printed with a plotter. Also the PSCS display can be reset to restart the display process.

PSCS Example Rules:

Rule 1
IF no blocks exist
THEN
generate a system controller.

Rule 2
IF a state exists which has a macro AND
this macro has not been mapped to a block
THEN
find a corresponding macro in the library
and generate a block for this macro.

Rule 3
IF there is a transition between two
states AND there are macros in these
states using the same argument
THEN
make a connection from a register
corresponding to the first macro to
another register corresponding to the
second macro.

Rule 4
IF a register has only a single connection
from another register
THEN
combine these registers into
a single register.

Rule 5
IF there are two comparators AND
input data widths are of the same size AND
one input of these is same AND
the outputs of the comparators are used to
perform the same operation.
THEN
combine these comparators into
a single comparator.

Rule 6
IF there is a data without a register
THEN
allocate a register for this data.

Rule 7
IF all the blocks have been interconnected AND a block has a few terminals not connected
THEN
remove the block and its terminals, or issue an error message.

Rule 8
IF memory is to be used, but a block has not been created for it
THEN
create a memory block with data, address, read and write data and control terminals.

Rule 9
IF a register has a single connection to a counter
THEN
combine the register and the counter; remove the register and its terminals.

Rule 10
IF there are connections to a terminal of a block from many different blocks
THEN
insert a multiplexor;
remove the connections to the terminals and connect them to the input of the multiplexor;
connect the output of the multiplexor to the input of the block.

Additional rules address the following points:

- remove cell(s) that can be replaced by using the outputs of other cell(s)
- reduce multiplexor trees
- use fan-out from the cells, etc.

Soft Drink Vending Machine Controller Design Example

The following example illustrates how the previously described features of the present invention are employed in the design of an application specific integrated circuit (ASIC). In this illustrative example the ASIC is designed for use as a

vending machine controller. The vending machine controller receives a signal each time a coin has been deposited in a coin receiver. The coin value is recorded and when coins totalling the correct amount are received, the controller generates a signal to dispense a soft drink. When coins totalling more than the cost of the soft drink are received, the controller dispenses change in the correct amount.

This vending machine controller example is patterned after a textbook example used in teaching digital system controller design. See Fletcher, William I., An Engineering Approach to Digital Design, Prentice-Hall, Inc., pp. 491-505. Reference may be made to this textbook example for a more complete explanation of this vending machine controller requirements, and for an understanding and appreciation of the complex design procedures prior to the present invention for designing the hardware components for a controller.

Figure 10 illustrates a flowchart for the vending machine controller system. This flowchart would be entered into the KBSC system by the user through the flowchart editor. Briefly reviewing the flowchart, the controller receives a coin present signal when a coin is received in the coin receiver. State0 and cond0 define a waiting state awaiting deposit of a coin. The symbol CP represents "coin present" and the symbol !CP represents "coin not present". State1 and cond1 determine when the coin has cleared the coin receiver. At state20, after receipt of a coin, the macro instruction ADD3.1 (lc, cv, sum) instructs the system to add lc (last coin) and cv (coin value) and store the result as sum. The macro instruction associated with state21 moves the value in the register sum to cv. The macro CMP.1 at state22 compares the value of cv with PR (price of

soft drink) and returns signals EQ, GT and LT. The condition cond2 tests the result of the compare operation CMP.1. If the result is "not greater than" (!GT.CMP.1), then the condition cond3 tests to see whether the result is "equal" (EQ.CMP.1). If the result is "not equal" (!EQ.CMP.1), then control is returned to state0 awaiting the deposit of another coin. If cond3 is EQ, then state4 generates a control signal to dispense a soft drink (droppop) and the macro instruction CLR.1(cv) resets cv to zero awaiting another customer.

If the total coins deposited exceed the price, then state30 produces the action "returncoin". Additionally, the macro DECR.1 (cv) reduces the value of cv by the amount of the returned coin. At state31 cv and PR are again compared. If cv is still greater than PR, then control passes to state30 for return of another coin. The condition cond5 tests whether the result of CMP.2 is EQ and will result in either dispensing a drink (droppop) true or branching to state0 awaiting deposit of another coin. The macros associated with the states shown in Figure 10 correspond to those defined in Table 1 above and define the particular actions which are to be performed at the respective states.

Appendix A shows the intermediate file or "statelist" produced from the flowchart of Figure 10. This statelist is produced as output from the EDSIM program 20 and is used as input to the PSCS program 30 (Figure 3).

Figure 11 illustrates for each of the macros used in the flowchart of Figure 10, the corresponding hardware blocks. It will be seen that the comparison macro CMP (A,B) results in the generation of a register for storing value A, a register for storing value B, and a comparator block and also produces control

paths to the system controller for the EQ, LT, and GT signals generated as a result of the comparison operation. The addition macro ADD (A,B,C) results in the generation of a register for each of the input values A and B, a register for the output value C, and in the generation of an adder block. The macro DECR (A) results in the generation of a counter block. The PSCS program 30 maps each of the macros used in the flowchart of Figure 10 to the corresponding hardware components results in the generation of the hardware blocks shown in Figure 12. In generating the illustrated blocks, the PSCS program 30 relied upon rules 1 and 2 of the above listed example rules.

Figure 13 illustrates the interconnection of the blocks of Figure 12 with data paths and control paths. Rule 3 was used by the data/control path synthesizer program 31 in mapping the data and control paths.

Figure 14 shows the result of optimizing the circuit by applying rule 4 to eliminate redundant registers. As a result of application of this rule, the registers R2, R3, R7, R8, and R9 in Figure 13 were removed. Figure 15 shows the block diagram after further optimization in which redundant comparators are consolidated. This optimization is achieved in the PSCS program 30 by application of rule 5.

Having now defined the system controller block, the other necessary hardware blocks and the data and control paths for the integrated circuit, the PSCS program 30 now generates a netlist 15 defining these hardware components and their interconnection requirements. From this netlist the mask data for producing the integrated circuit can be directly produced using available VLSI CAD tools.

Appendix A

```
name rpop ;

data path  @ic<0:5>, cv<0:5>, sum<0:5>, @pr<0:5>;

{

state4 : state0;
state30 : state31;
state21 : state22;
state20 : state21;

state0 :. !cp state0;
state0 :. cp state1;
state1 :. cp state1;
state1 :. !cp state20;
state22 :. GT.CMP.1 state30;
state22 :. !GT.CMP.1*EQ.CMP.1 state4;
state22 :. !GT.CMP.1*!EQ.CMP.1 state0;
state31 :. GT.CMP.2 state30;
state31 :. !GT.CMP.2*EQ.CMP.2 state4;
state31 :. !GT.CMP.2*!EQ.CMP.2 state0;

state30 :: returncoin;
state30 :: DECR.1(cv);
state4 :: droppop;
state4 :: CLR.1(cv);
state31 :: CMP.2(cv,pr);
state22 :: CMP.1(cv,pr);
state21 :: MOVE.1(sum,cv);
state20 :: ADD3.1(ic,cv,sum);

}
```

THAT WHICH I CLAIMED IS:

1. A computer-aided design system for designing an application specific integrated circuit directly from functional specifications for the integrated circuit, comprising

input means operable by a user for defining functional specifications for the integrated circuit, and

computer operated means for translating the functional specifications into a structural level definition of an integrated circuit.

2. The system as defined in Claim 1 wherein said functional specifications are comprised of a series of actions and conditions and wherein said structural level definition are comprised of blocks and interconnections between blocks.

3. The system as defined in Claim 1 wherein said input means comprises flow chart editor means for creating a flow chart having elements representing a series of actions and conditions which describe the functional specifications for the integrated circuit.

4. The system as defined in Claim 1 wherein said computer operated means comprises

a cell library defining a set of available integrated circuit hardware cells for performing actions and conditions, and

cell selection means for selecting from said cell library appropriate hardware cells corresponding to the blocks of said structural definition.

5. A computer-aided design system for designing an application specific integrated circuit directly from functional specifications for the integrated circuit, comprising

a macro library defining a set of possible actions and conditions;

input means operable by a user for defining functional specifications for the integrated circuit, said functional specifications being comprised of a series of actions and conditions, said input means including means to permit the user to specify for each action or condition in the defined series of actions and conditions a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing actions and conditions; and

cell selection means for selecting from said cell library for each macro specified by said specification means, appropriate hardware cells for performing the action or condition defined by the specified macro.

6. The system as defined in Claim 5 wherein said cell selection means comprises a knowledge Base containing rules for selecting hardware cells from said cell library and inference engine means for selecting appropriate hardware cells from said cell library in accordance with the rules of said knowledge base.

7. The system as defined in Claim 5 wherein said input means comprises flowchart editor means for creating a flowchart having elements representing said series of actions and conditions.

8. The system as defined in Claim 7 additionally including flowchart simulator means for simulating the functions defined in the flowchart to enable the user to verify the operation of the integrated circuit.

9. The system as defined in Claim 5 wherein said input means comprises means for receiving user input of a list defining the series of actions and conditions.

10. The system as defined in Claim 5 additionally including data path generator means cooperating with said cell selection means for generating data paths for the hardware cells selected by said cell selection means.

11. The system as defined in Claim 10 wherein said data path generator means comprises a knowledge base containing rules for selecting data paths between hardware cells and inference engine means for selecting data paths between the hardware cells selected by said cell selection means in accordance with the rules of said knowledge base and the arguments of the specified macros.

12. The system as defined in Claim 10 additionally including control generator means for generating a controller and control paths for the hardware cells selected by said cell selection means.

13. The system as defined in Claim 5 including netlist generator means cooperating with said cell selection means for generating as output from the system a netlist defining the hardware cells which are needed to achieve the functional requirements of the integrated circuit and the connections therebetween.

14. The system as defined in Claim 13 additionally including mask data generator means for generating from said netlist the mask data required to produce an integrated circuit having the specified functional requirements.

15. A computer-aided design system for designing an application specific integrated circuit directly from a flowchart defining the functional requirements of the integrated circuit, comprising

a macro library defining a set of possible actions and conditions;

flowchart editor means operable by a user for creating a flowchart having elements representing actions and conditions;

said flowchart editor means including macro specification means for permitting the user to specify for each action or condition represented in the flowchart a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing actions and conditions;

cell selection means for selecting from said cell library for each specified macro, appropriate hardware cells for performing the action or condition defined by the specified macro; and

data path generator means cooperating with said cell selection means for generating data paths for the hardware cells selected by said cell selector means.

16. The system as defined in Claim 15 wherein said cell selection means comprises a knowledge base containing rules for selecting hardware cells from said cell library and inference engine means for selecting appropriate hardware cells from said cell library in accordance with the rules of said knowledge base, and wherein said data path generator means comprises a knowledge base containing rules for selecting data paths between hardware cells and inference engine means for selecting data paths between hardware cells selected by said cell selection means in accordance with the rules of said knowledge base and the arguments of the specified macros.

17. The system as defined in Claim 15 additionally including control generator means for generating a controller and control paths for the hardware cells selected by said cell selection means.

18. A computer-aided design system for designing an application specific integrated circuit directly from a flowchart defining the functional requirements of the integrated circuit, comprising

flowchart editor means operable by a user for creating a flowchart having boxes representing actions, diamonds representing conditions, and lines with arrows representing transitions between actions and conditions and including means for specifying for each box or diamond, a particular action or condition to be performed;

a cell library defining a set of available integrated circuit hardware cells for performing actions and conditions;

a knowledge base containing rules for selecting hardware cells from said cell library and for generating data and control paths for hardware cells; and

expert system means operable with said knowledge base for translating the flowchart defined by said flowchart editor means into a netlist defining the necessary hardware cells and data and control paths required in an integrated circuit having the specified functional requirements.

19. The system as defined in claim 18 including mask data generator means for generating from said netlist the mask data required to produce an integrated circuit having the specified functional requirements.

20. A computer-aided design process for designing an application specific integrated circuit which will perform a desired function comprising

storing a set of definitions of possible actions and conditions;

storing data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set;

describing for a proposed application specific integrated circuit a series of actions and conditions;

specifying for each described action and condition of the series one of said stored definitions which corresponds to the desired action or condition to be performed; and

selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit.

21. A process as defined in Claim 20 wherein said step of selecting a hardware cell comprises applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in a knowledge base.

22. A process as defined in Claim 21 including the further step of generating data paths for the selected integrated circuit hardware cells.

23. A process as defined in Claim 22 wherein said step of generating data paths comprises applying to the selected cells a set of data path rules stored in a knowledge base and generating the data paths therefrom.

24. A process as defined in Claim 23 including the further step of generating control paths for the selected integrated circuit hardware cells.

25. A process as defined in Claim 20 including the further step of generating for the selected integrated circuit hardware cells a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.

26. A process as defined in Claim 25 including generating from the netlist the mask data required to produce an integrated circuit having the desired function.

27. A knowledge based design process for designing an application specific integrated circuit which will perform a desired function comprising

storing in a macro library a set of macros defining possible actions and conditions;

storing in a cell library a set of available integrated circuit hardware cells for performing the actions and conditions;

storing in a knowledge base a set of rules for selecting hardware cells from said cell library to perform the actions and conditions defined by the stored macros;

describing for a proposed application specific integrated circuit a series of actions and conditions which carry out the function to be performed by the integrated circuit;

specifying for each describing action and condition of said series a macro selected from the macro library which corresponds to the action or condition; and

applying rules of said knowledge base to the specified macros to select from said cell library the hardware cells required for performing the desired function of the application specific integrated circuit.

28. A process as defined in Claim 27 wherein said step of describing a series of actions and conditions comprises creating a flowchart comprised of elements representing actions, architecture independent and conditions.

29. A process as defined in Claim 27 also including the steps of

storing in said knowledge base a set of rules for creating data paths between hardware cells, and

applying rules of said knowledge base to the specified means to create data paths for the selected hardware cells.

30. A process as defined in Claim 29 also including the steps of generating a controller and generating control paths for the selected hardware cells.

KNOWLEDGE BASED METHOD AND APPARATUS
FOR DESIGNING INTEGRATED CIRCUITS
USING FUNCTIONAL SPECIFICATIONS

Abstract of the Disclosure

The present invention provides a computer-aided design system and method for designing an application specific integrated circuit which enables a user to define functional specifications for the integrated circuit and which translates the functional specifications into the detailed information needed for directly producing the integrated circuit. The functional specifications of the desired integrated circuit can be defined at the functional level in a flowchart format. From the flowchart, the system and method uses artificial intelligence and expert systems technology to generate a system controller, to select the necessary integrated circuit hardware cells needed to achieve the functional specifications, and to generate data and control paths for operation of the integrated circuit. This list of hardware cells and their interconnection requirements is set forth in a netlist. From the netlist it is possible using known manual techniques or existing VLSI CAD layout systems to generate the detailed chip level topological information (mask data) required to produce the particular application specific integrated circuit.

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

Attorney Docket: 3868-2

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS,

the specification of which

(check one)

[X] is attached hereto.

[ ] was filed on _____ as

Application Serial No. _____

and was amended on _____
(if applicable)

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, Section 1.56(a).

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

English Language Declaration

Prior Foreign Application(s)			Priority Claimed	
None (Number)	(Country)	(Day/Month/Year Filed)	[ ] Yes	[ ] No
(Number)	(Country)	(Day/Month/Year Filed)	[ ] Yes	[ ] No
(Number)	Country)	(Day/Month/Year Filed)	[ ] Yes	[ ] No

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

None (Appln. Serial No.)	(Filing Date)	(Status) (patented/pending/aban.)
(Appln. Serial No.)	(Filing Date)	(Status) (patented/pending/aban.)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

English Language Declaration

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith (list name and registration number).

Raymond O. Linker, Jr.
Registration No. 26,419

Send correspondence to: Raymond O. Linker, Jr.
Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, North Carolina 28234

Direct telephone calls to: (name and telephone number)

Raymond O. Linker, Jr.
(704) 377-1561

Full name of sole inventor: Hideaki Kobayashi

Inventor's Signature: [signature] Date: January 12, 1988

Residence: Columbia, South Carolina

Citizenship: Japan

Post Office Address: 1401 Main Street
Columbia, South Carolina 29201

Full name of second inventor: Masahiro Shindo

Inventor's Signature: [signature] Date: Dec. 26. 1987

Residence: Osaka, Japan

Citizenship: Japan

Post Office Address: 13-1, Himemuro-cho
Ikeda, Osaka, 563
Japan

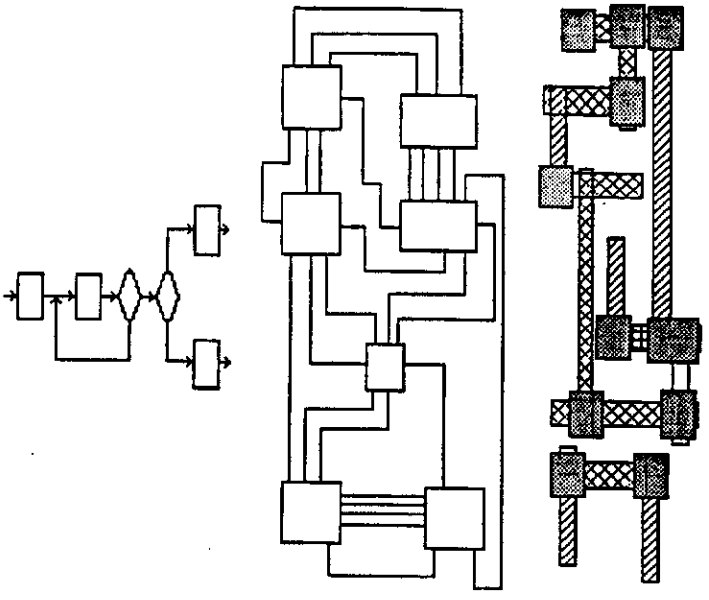


Figure 1a
Functional Level

Figure 1b
Structural Level

Figure 1c
Physical Layout Level

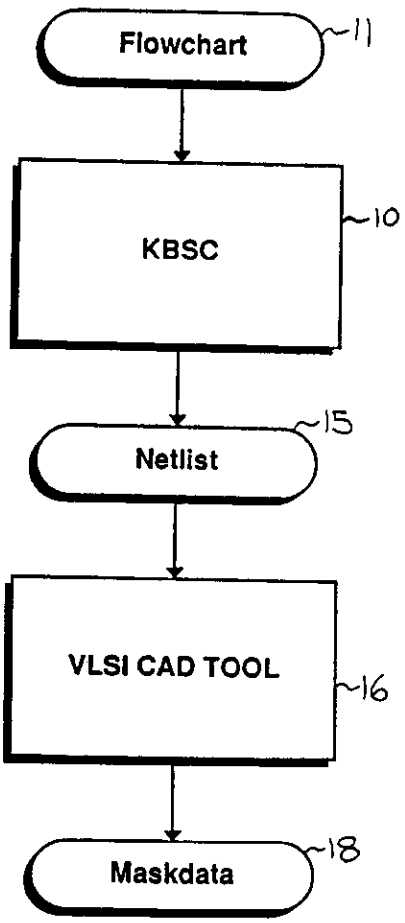


Figure 2

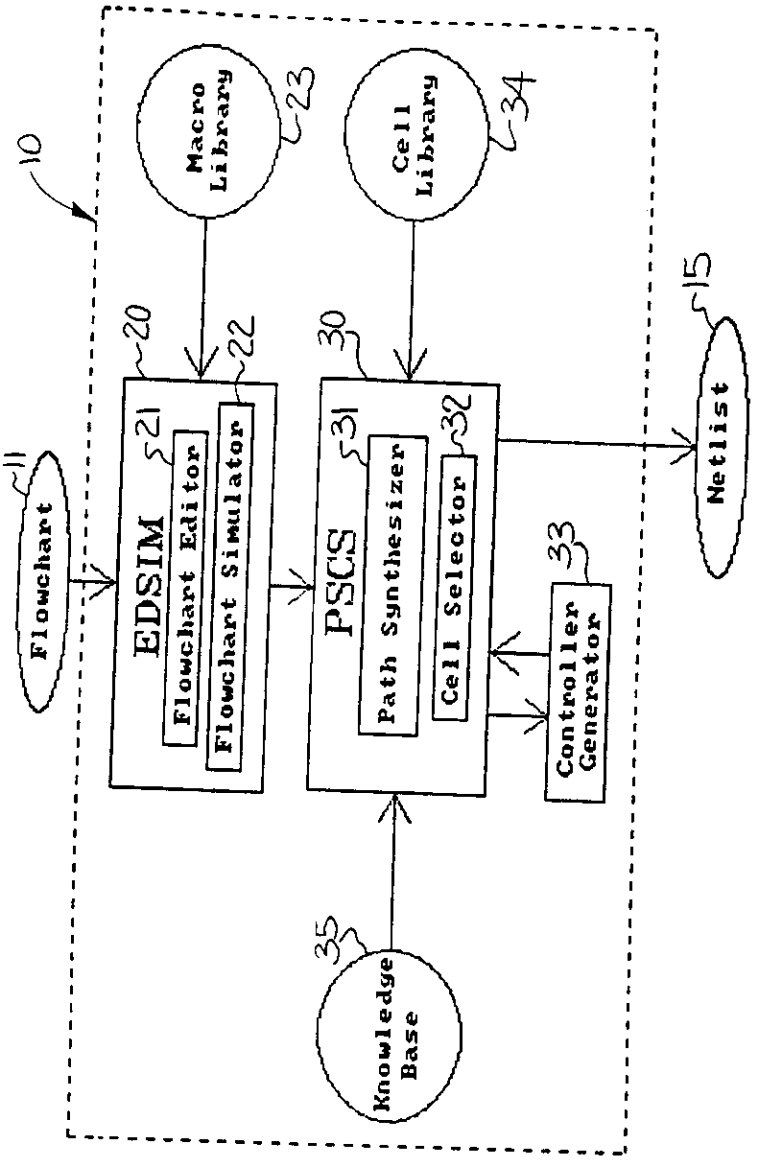

10
Macro Library
23
Cell Library
34
15
20
22
30
Netlist
11
EDSIM
21
Flowchart Editor
Flowchart Simulator
PSCS
31
Path Synthesizer
32
Cell Selector
33
Controller Generator
Flowchart
35
Knowledge Base

Figure 3.

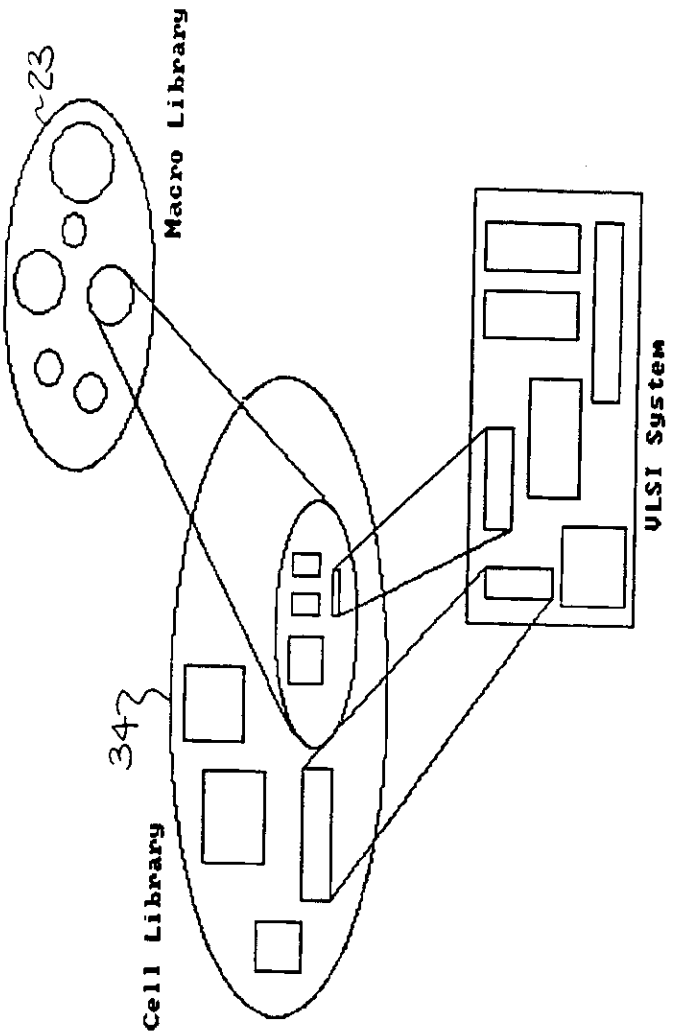


Figure 4.

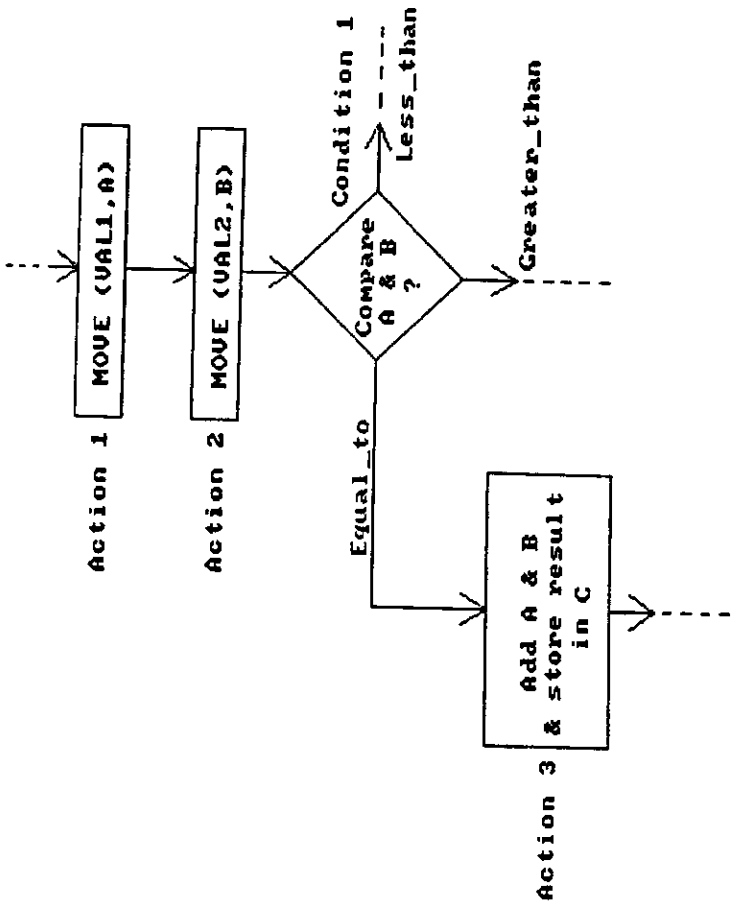

MOVE (VAL1,A)
MOVE (VAL2,B)
Compare A & B ?
Condition 1
Less_than
Greater_than
Equal_to
Add A & B & store result in C
Action 1
Action 2
Action 3

Figure 5

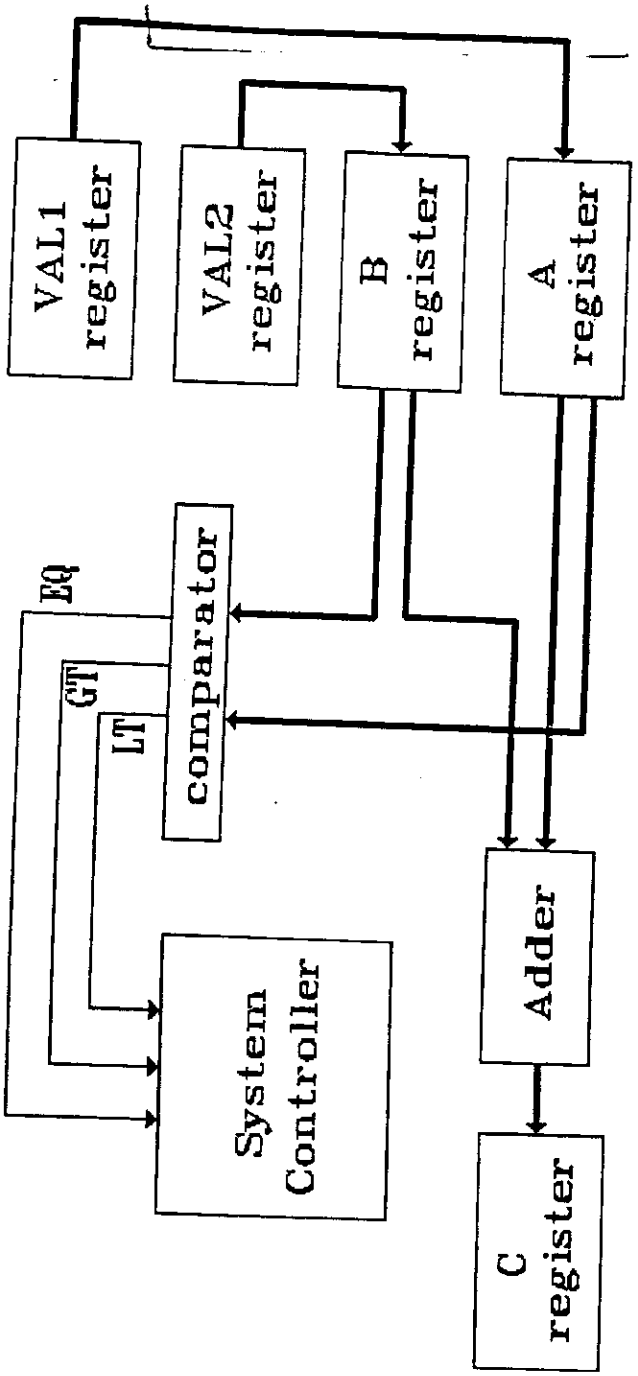

VAL1 register
VAL2 register
B register
A register
comparator
EQ
GT
LT
System Controller
Adder
C register

Figure 6

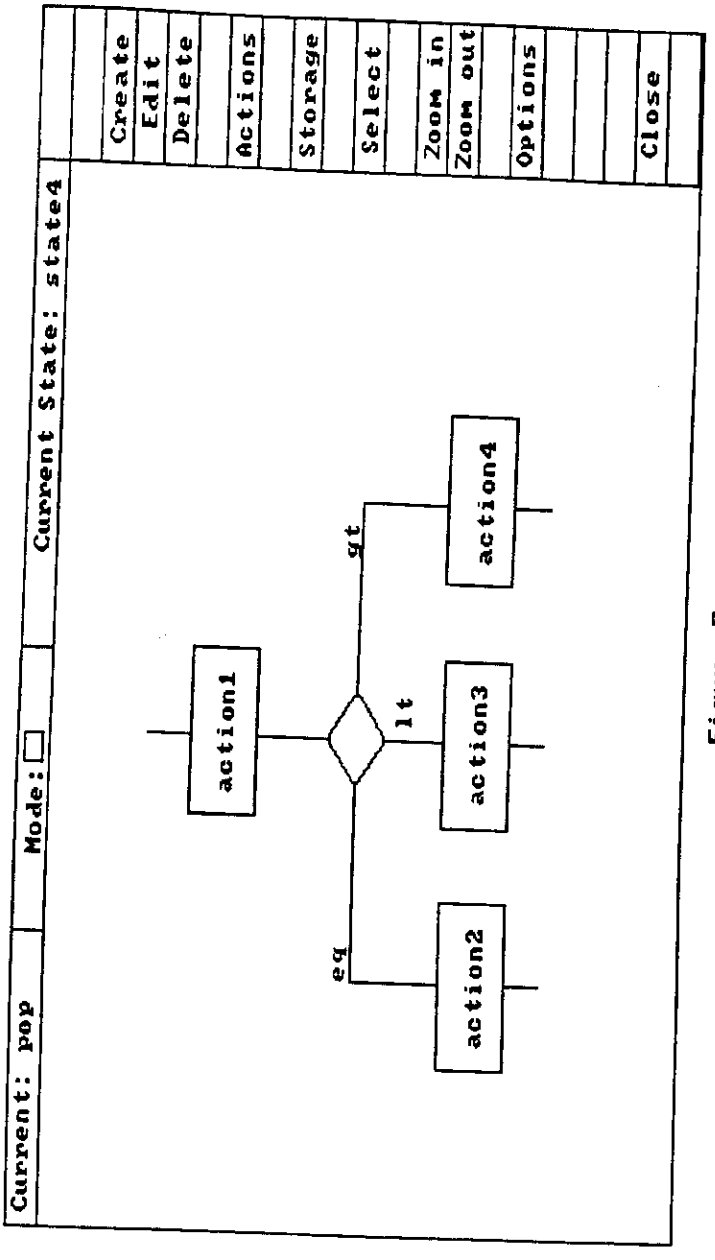

Create
Edit
Delete
Actions
Storage
Select
Zoom in
Zoom out
Options
Close
Current State: state4
Mode:
Current: pop
action1
gt
action4
lt
action3
eq
action2

Figure 7

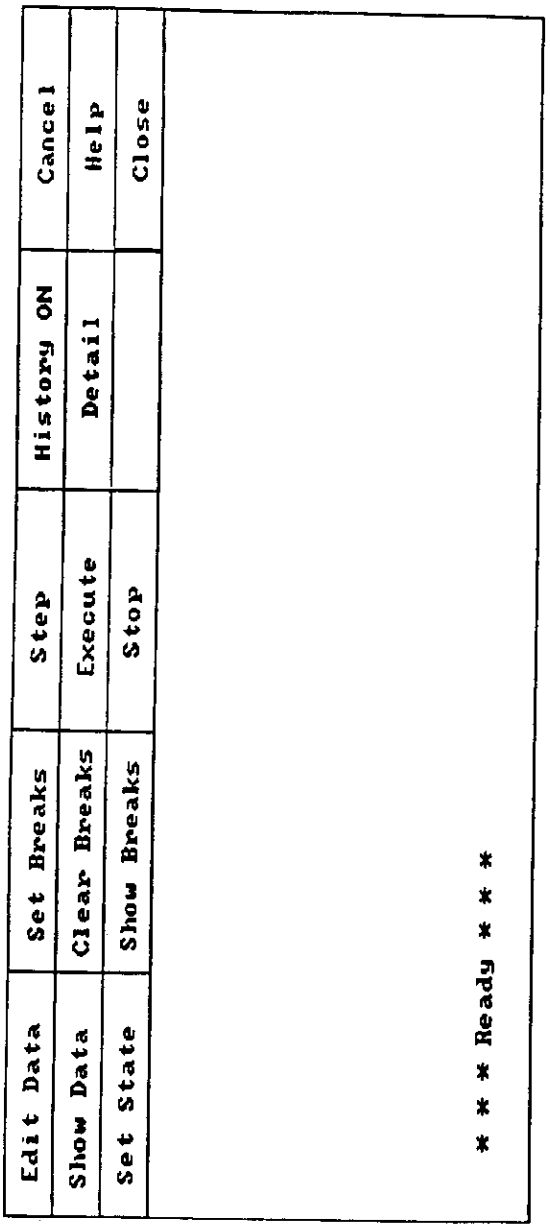

Edit Data
Show Data
Set State
Set Breaks
Clear Breaks
Show Breaks
Step
Execute
Stop
History ON
Detail
Cancel
Help
Close
* * * Ready * * *

Figure 8

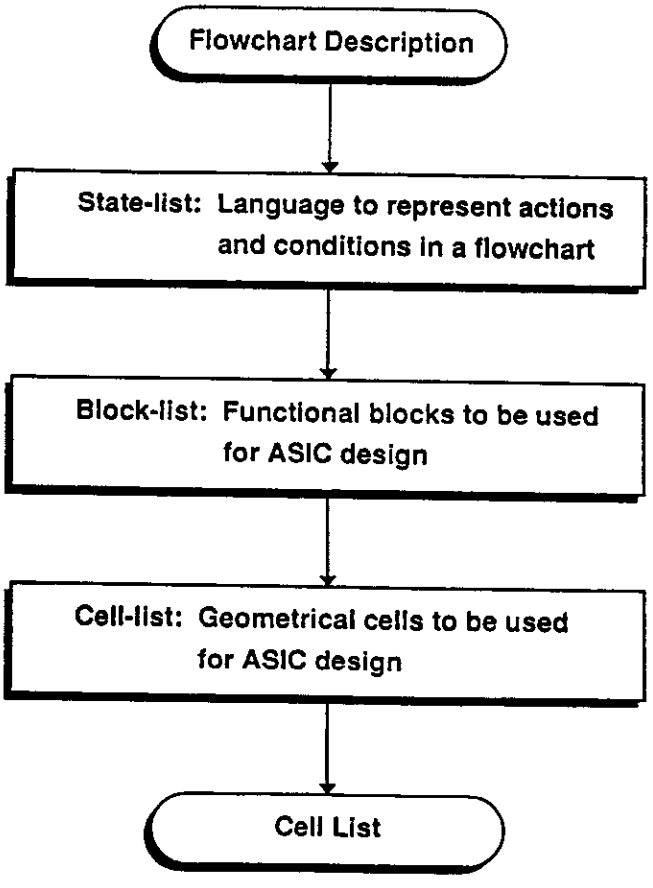

Flowchart Description
State-list: Language to represent actions and conditions in a flowchart
Block-list: Functional blocks to be used for ASIC design
Cell-list: Geometrical cells to be used for ASIC design
Cell List

Figure 9

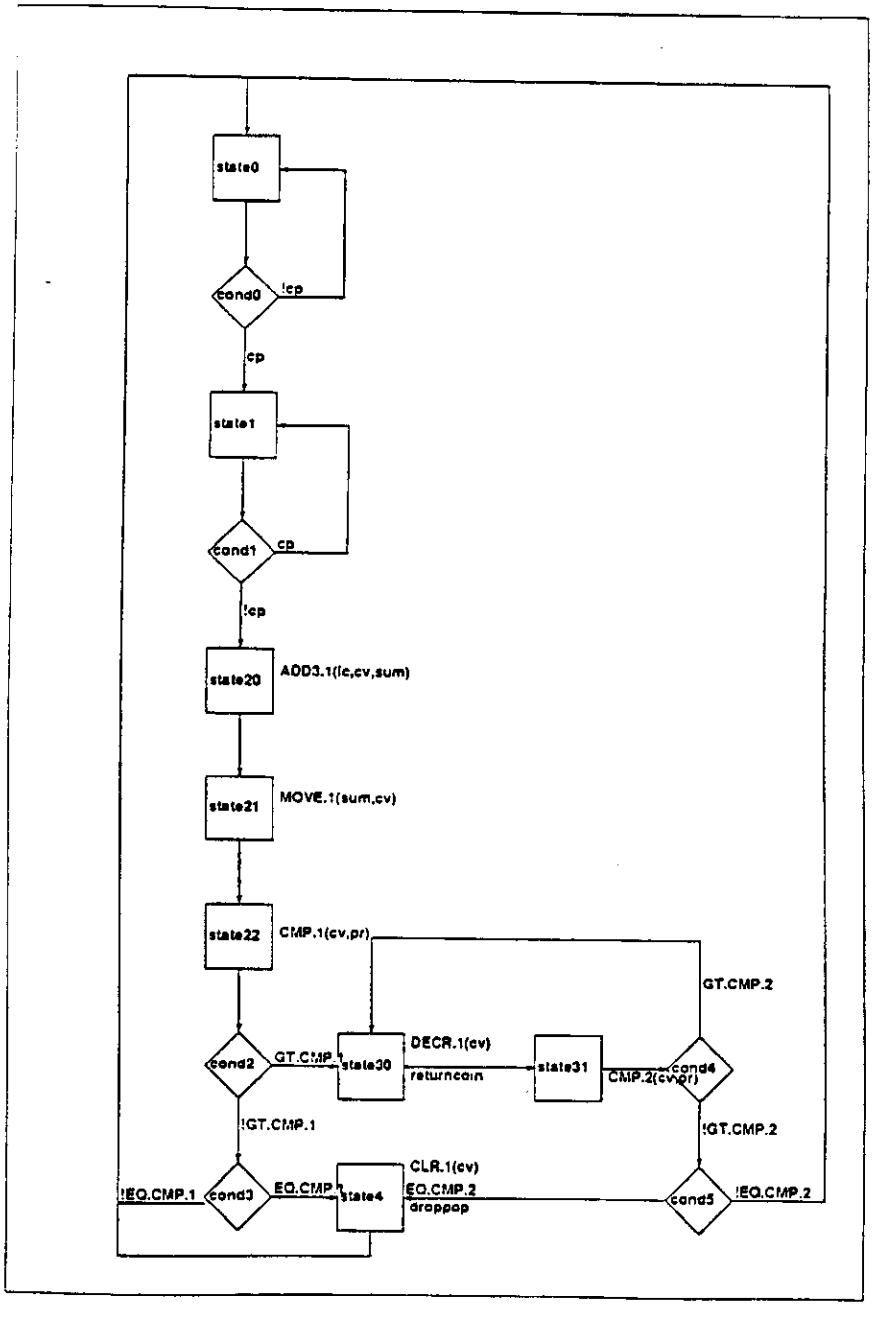

state0
cond0
!cp
cp
state1
cond1
cp
!cp
state20
ADD3.1(ic,cv,sum)
state21
MOVE.1(sum,cv)
state22
CMP.1(cv,pr)
GT.CMP.2
cond2
GT.CMP.1
state30
DECR.1(cv)
returncoin
state31
cond4
CMP.2(cv,pr)
!GT.CMP.1
!GT.CMP.2
CLR.1(cv)
!EQ.CMP.1
cond3
EQ.CMP.1
state4
EQ.CMP.2
droppop
cond5
!EQ.CMP.2

Figure 10

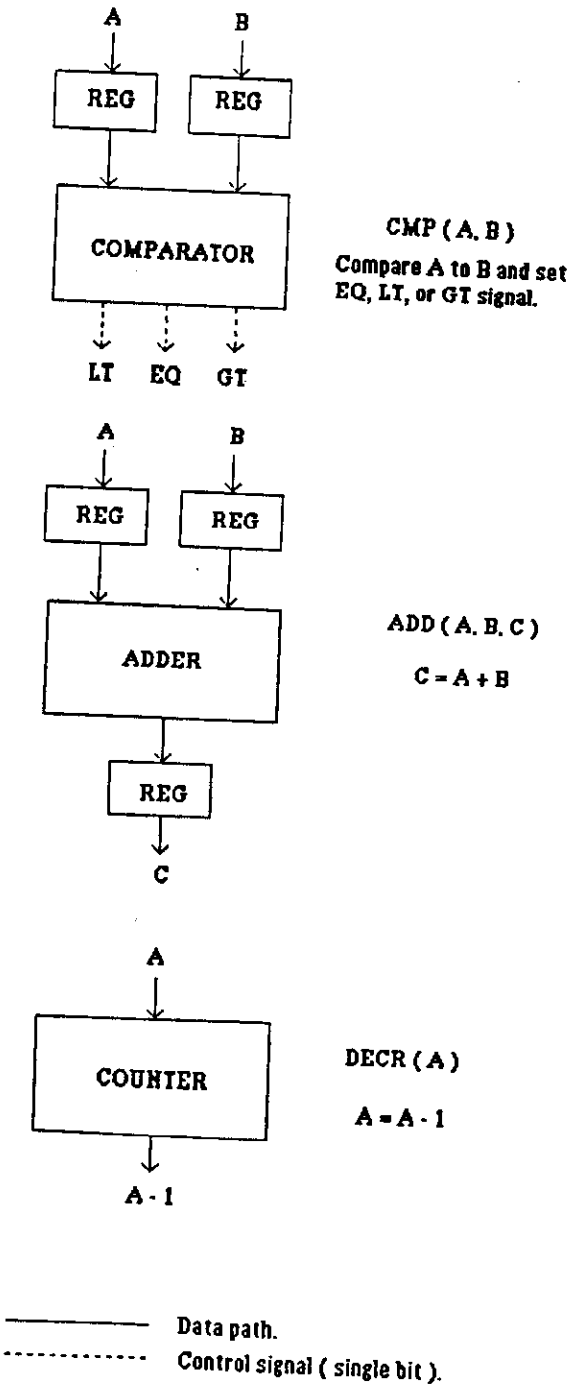

A
B
REG
REG
COMPARATOR
LT
EQ
GT
CMP (A, B)
Compare A to B and set EQ, LT, or GT signal.
A
B
REG
REG
ADDER
REG
C
ADD (A, B, C)
C = A + B
A
COUNTER
A - 1
DECR (A)
A = A - 1
Data path.
Control signal (single bit).

Figure 11

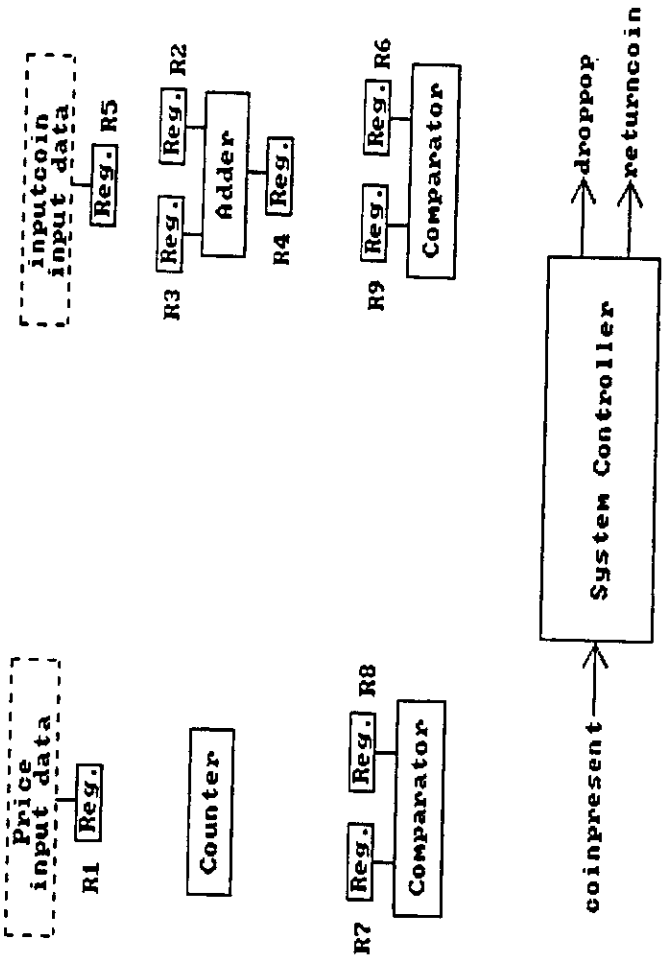

inputcoin
input data
Reg.
R5
Reg.
R2
Reg.
Adder
R3
Reg.
R4
Reg.
R6
Reg.
Comparator
R9
Reg.
droppop
returncoin
System Controller
Price
input data
Reg.
R1
Counter
Reg.
R8
Reg.
Comparator
R7
Reg.
coinpresent

Figure 12

As Original Filed

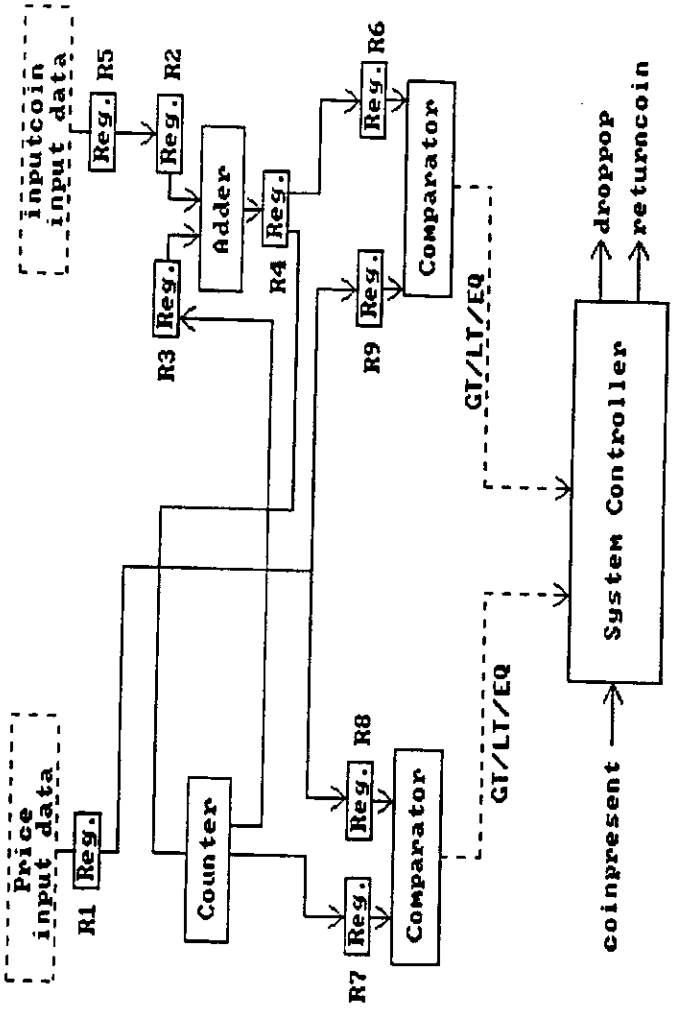


Figure 13

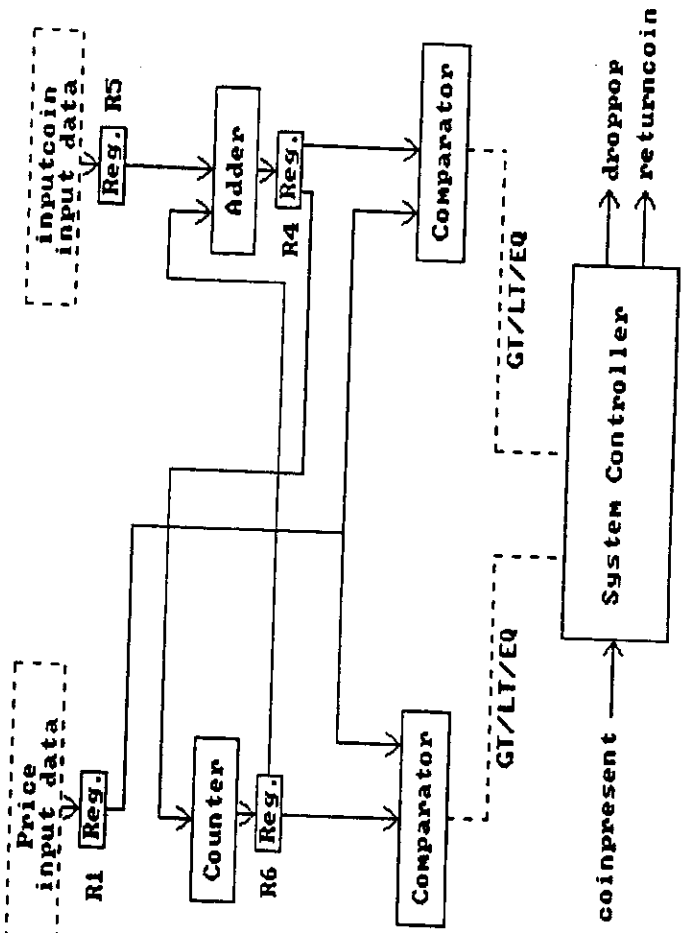

Inputcoin
input data
Reg.
R5
Adder
Reg.
R4
Comparator
GT/LT/EQ
dropcoin
returncoin
System Controller
Price
input data
Reg.
R1
Counter
Reg.
R6
Comparator
GT/LT/EQ
coinpresent

Figure 14

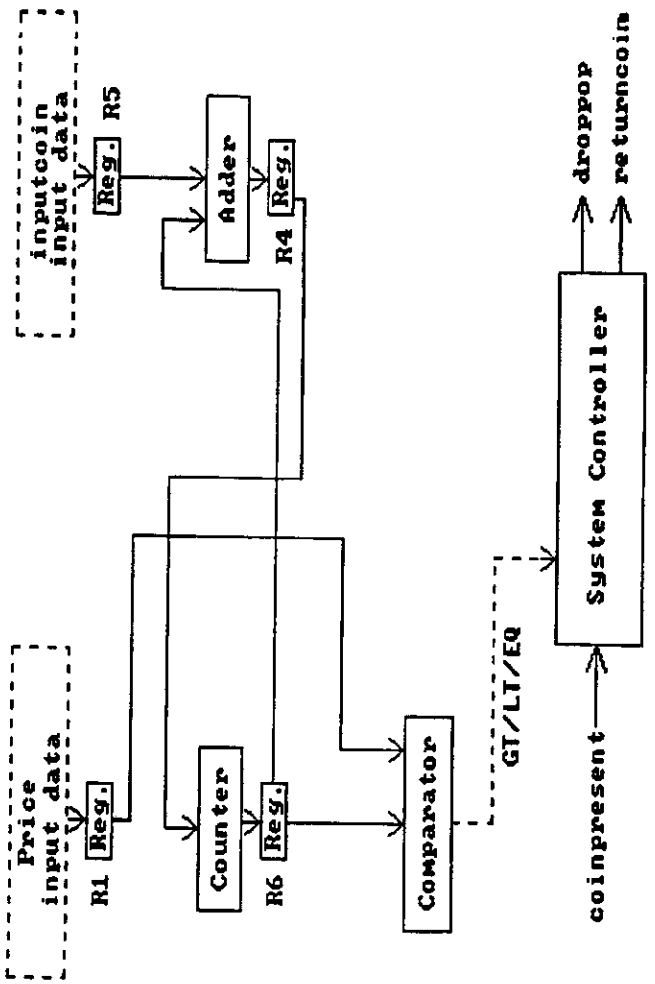

inputcoin
input data
Reg.
R5
Adder
Reg.
R4
dropcoin
returncoin
System Controller
Price
input data
Reg.
R1
Counter
Reg.
R6
Comparator
GT/LT/EQ
coinpresent

Figure 15

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Hideaki Kobayashi and
Masahiro Shindo
Filed: Concurrently Herewith
For: KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS

The Honorable Commissioner of
Patents and Trademarks
Washington, D.C. 20231

January 13, 1988

INFORMATION DISCLOSURE STATEMENT

Sir:

The patents listed on the attached form PTO 1449 were considered by applicants and applicants' attorney in the preparation of this application. While these patents may be of some general interest as background information in the field of artificial intelligence technology, they are not believed to be material to the patentability of the present invention. They are submitted herewith for consideration by the Examiner pursuant to Rule 60.

The excerpt from the textbook An Engineering Approach to Digital Design is mentioned in applicants' specification. A copy is enclosed for consideration by the Examiner and to complete the record.

Respectfully submitted,

Raymond O. Linker, Jr.
Registration No. 26,419

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, North Carolina 28234
(704) 377-1561
ROLjr:klc
Our File 3868-2

Sheet 1 of 1

FORM PTO-1449 (REV. 7-80) U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

ATTY. DOCKET NO. 3868-2

SERIAL NO. 07/143,821

APPLICANT Hideaki Kobayashi

FILING DATE Concurrently herewith

GROUP 234

LIST OF PRIOR ART CITED BY APPLICANT

(Use several sheets if necessary)

MAIL ROOM JAN 13 1988 U.S. PATENT & TRADEMARK OFFICE

U.S. PATENT DOCUMENTS

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILING DATE IF APPROPRIATE
VT	AA	4 6 4 8 0 4 4	3-3-87	Hardy, et al.	364	513	
VT	AB	4 6 5 8 3 7 0	4-14-87	Erman, et al.	364	513	
VT	AC	4 6 7 5 8 2 9	6-23-87	Clemenson	364	513	
	AD						
	AE						
	AF						
	AG						
	AH						
	AI						
	AJ						
	AK						

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION YES	TRANSLATION NO
	AL							
	AM							
	AN							
	AO							
	AP							

OTHER PRIOR ART *(Including Author, Title, Date, Pertinent Pages, Etc.)*

VT	AR	An Engineering Approach to Digital Design, William I. Fletcher, Prentice-Hall, Inc., pp. 491-505.
	AS	14
	AT	

EXAMINER V TRANS

DATE CONSIDERED 11/18/88

RCL000065

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

USCOMM-DC 60-3966

EXHIBIT 5
(Part 2 of 4)

TO: Mr. Linker
FROM: Dr. Kobayashi / ICC
12/22/87

AN ENGINEERING APPROACH TO DIGITAL DESIGN

WILLIAM I. FLETCHER
Utah State University
Logan, Utah

PRENTICE-HALL, INC., *Englewood Cliffs, New Jersey 07632*

FLAG=1 and $\overline{CLK}$, all of which can be traced through the detailed flow diagram.

Now consider one more design problem illustrating the use of the design phases (1–9). This problem requires you to interpret a set of detailed specifications and further illustrate the functionality of the system controller concept. Special thanks to Prof. Richard Ohran of Brigham Young University for suggesting a problem such as this back in 1971.

EXAMPLE 7-5: The Pop Machine Controller

Introduction and specifications. The El-Rip-O Vending Machine Company wishes to update its Model 1909 mechanical vending machine. It is desired that a first generation digital controlled prototype system be developed for test and evaluation; nothing really fancy, just an evaluation prototype. The El-Rip-O Company has entered into an OEM (Original Equipment Manufacturers) agreement with the Futzel–Boopsdink Pipe and Die Company to provide the coin receiver, coin changer, and pop dropping systems for the new El-Rip-O Model 1971 system when it is fully developed.

Preliminary specifications. The initial digital control system should be developed such that it will direct the control of the coin receiver, coin changer, and pop drop mechanics and provide the El-Rip-O Company with a system capable of automatically dispensing soda pop at 30¢ per can and making the proper change retrieval for the following coin sequences of nickels, dimes quarters, and half-dollars.

Prescribed operation. Upon the insertion of each coin, the controller is to record the coin value and issue the proper change, if required, in nickels only, then drop a can of soda pop. At this time a solenoid is to be activated that lets the coins collected in the coin sequence drop into a common collector box. This system is to have a coin release feature for the manual release of jammed coins and coins collected in "short" sequences, and the coin release feature is also intended to reset the system to an initial condition.

Coin receiver specification. The Futzel–Boopsdink Pipe and Die Model CR-1971 coin receiver is described by the following features:

(1) single slot coin entry;
(2) electronic coin detection;
(3) guaranteed coin detection for U.S. half dollars, quarters, dimes, and nickels;
(4) automatic nonvalid and bent coin rejection;
(5) all control inputs and outputs are standard TTL compatible;
(6) coin-catching mechanism and manual coin release feature;
(7) a special mechanical mechanism that prevents coin over-run.

Electrical specifications. See Figure 7-32.

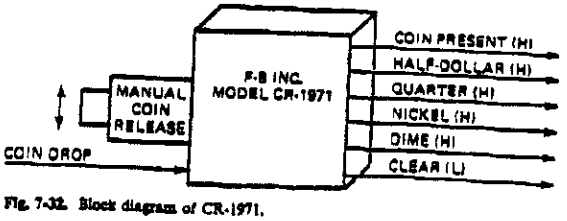


Fig. 7-32. Block diagram of CR-1971.

Signal description.

COIN DROP—an input that commands the CR-1971 to drop collected coins into the common collection box.

CLEAR—an output that will go to a 5 volt ±0.5 level for the duration of the depression of the coin release.

COIN PRESENT—an output signifying that a coin is present in the coin receiver and denomination has been determined.

HALF DOLLAR, QUARTER, DIME and NICKEL—outputs signifying the denomination of coin present.

The timing relationship between COIN PRESENT, HALF-DOLLAR, QUARTER, DIME, and NICKEL is described by the timing diagram shown in Figure 7-33.

Coin changer. The Futzel–Boopadink Pipe and Die Model CC-1971 coin changer (see Figure 7-34), developed by the Stuck–Muzzey subsidiary, features:

(1) fast, reliable, electro-mechanical nickel-ejecting system (100 msec per nickel maximum);
(2) special READY status output which indicates when another coin ejection sequence can be started;
(3) automatic load of 50 nickels reserve.

EJECT NICKEL— TTL compatible input, pulse triggered with minimum width of 10 msec restriction.

CHANGER READY—an output status line indicating the changer will respond to EJECT NICKEL.

Timing relations. See Figure 7-35.

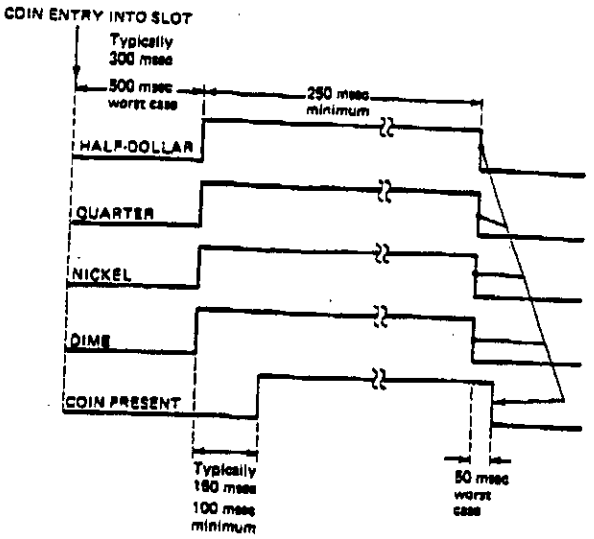


Fig. 7-33. Timing detail of the CR-1971.

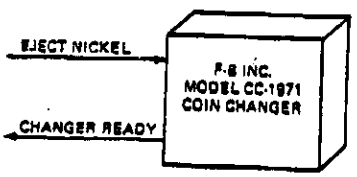


Fig. 7-34. The block diagram of the CC-1971.

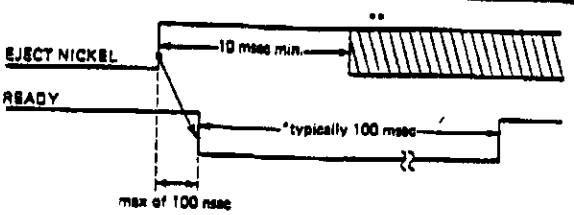


Fig. 7-35. Timing detail of the CC-1971.

Pop-drop mechanism The Futzel-Boopsdink Model PD-1971 pop-drop mechanism developed by Bend-A-Can Division (see Figure 7-36) is a fast solenoid operated device featuring:

(1) single TTL compatible drop command input;
(2) pulse operated with 10 msec minimum pulse width;
(3) status READY line

Timing and operation is identical to that of the coin changer.

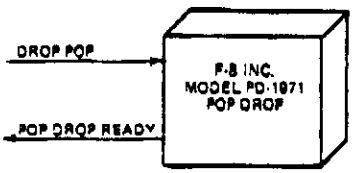


Fig. 7-36. The block diagram of the PD-1971. Note that POP DROP READY will not return to the READY status if the mechanism is jammed.

7-11.3 SYSTEM DEVELOPMENT

First let it be noted that the system to be designed has certain frailties related to its cheating the customer as well as the customer cheating it. However, since this development is only an evaluation prototype, these frailties can be temporarily overlooked. The specifications for a foolproof system are given as a design exercise and left for you to work out the fine details.

Phase I. Because of your intimate knowledge of vending machines derived from the everyday encounters with these beasts (see Figure 7-37), along with the specifications given, Phase I is pretty well covered.

Fig. 7-37. Man and the vending machine.

Phase II. From the knowledge accounted for in Phase I, the bare-bones block diagram as shown in Figure 7-38 can be readily developed. This bare-bones block diagram is supported by the first-cut flow diagram shown in Figure 7-39. Note that this flow diagram clearly defines the proposed big picture sequential behavior of the controller operation. Now keep in mind that this proposed operation is but one of several possible alternatives. However, it was selected on the basis of the following criteria:

(1) It provides a practical and effective system.
(2) The author feels that it nicely demonstrates the logical sequence of events.
(3) It also demonstrates how properly designed "hard wired" systems can have a reasonable degree of flexibility.

Though these criteria are somewhat biased, the concept of making a selection from several possible alternatives should not be considered as such. It is very important that you derive several alternatives and then select one of these based on some criteria. Now these criteria can vary depending on the design requirements, but the major cornerstones should be:

(1) Practicality
(2) Effectiveness
(3) Efficiency
(4) Degree of flexibility

The flexibility facet should be treated in the light of possible future applications and modifications. This is in keeping with the invariable fact that somebody will want something changed at the most inopportune time. However, don't let these anticipations completely override your preliminary preparations; just keep them in mind. For example, though there is no mention of price changes or multi-pop selection requirements in the specifications for the pop machine, these features are included with no real extra effort or cost. They are simply a natural fallout of the proposed system.

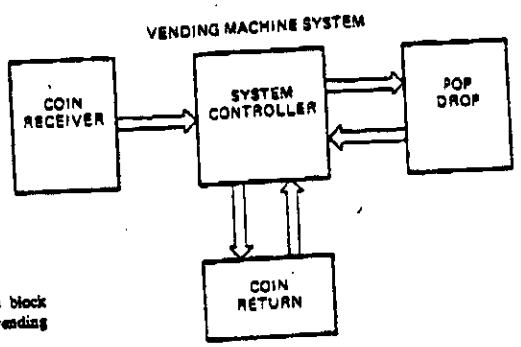


Fig. 7-38. The bare-bones block diagram of the prototype vending machine.

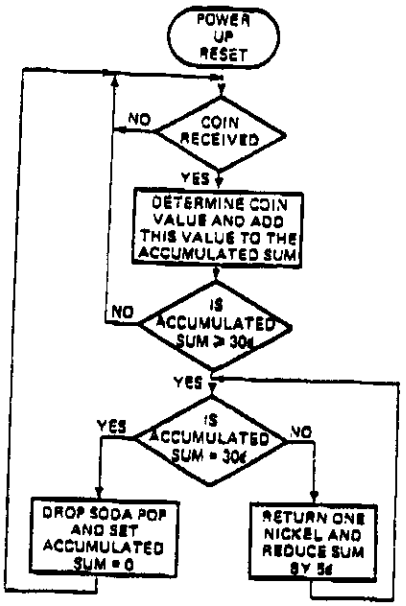


Fig. 7-39. A proposed first-cut flow diagram defining the vending machine's basic operation.

7-11.4 THE DETAILED FLOW DIAGRAM, FUNCTIONAL PARTITION AND MDS DIAGRAM DESCRIPTION

The detailed flow diagram shown in Figure 7-40 defines the exact sequential behavior of the functional partition shown in Figure 7-41. It should be clear that this flow diagram is an extension to the first-cut diagram. Along with this it also defines the timing relations of the control signal to and from the subsystems. The functional partition typifies the system controller in its presiding role. It should be noted that the arithmetic operations called for are carried out by the interconnection of the ADDER, PIPO REGISTER, DOWN COUNTER, and COMPARATOR. The PIPO REGISTER was added to insure the edge-triggered loading of the COUNTER. This register/counter complex is referred to as the ACCUMULA-

TOR (ACC) in this example. Further, it should be noted that a "strappable" compare word allows for easy price changing up to 75¢. The value of each coin is then encoded to the binary value of the number of nickels it represents in accordance with the indicated encoding (w_4, w_3, w_2, w_1) specification table. This encoded value is then added to the present value of the accumulator on the rising-edge of COIN PRESENT. Thus the basic functions called for by the first-cut and detailed flow diagram are brought to light with a hardware description. Note further that the IC package count is up to six packages, all with 16 pins or less.

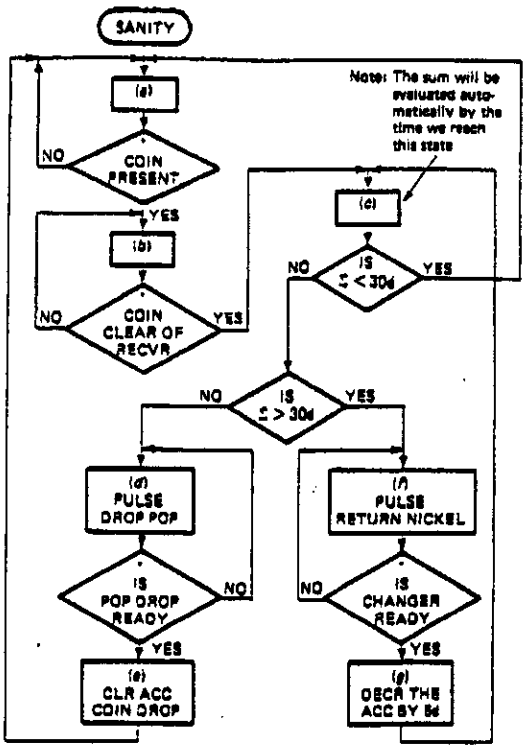


Fig. 7-48. The detailed flow diagram for the prototype pop machine system controller.

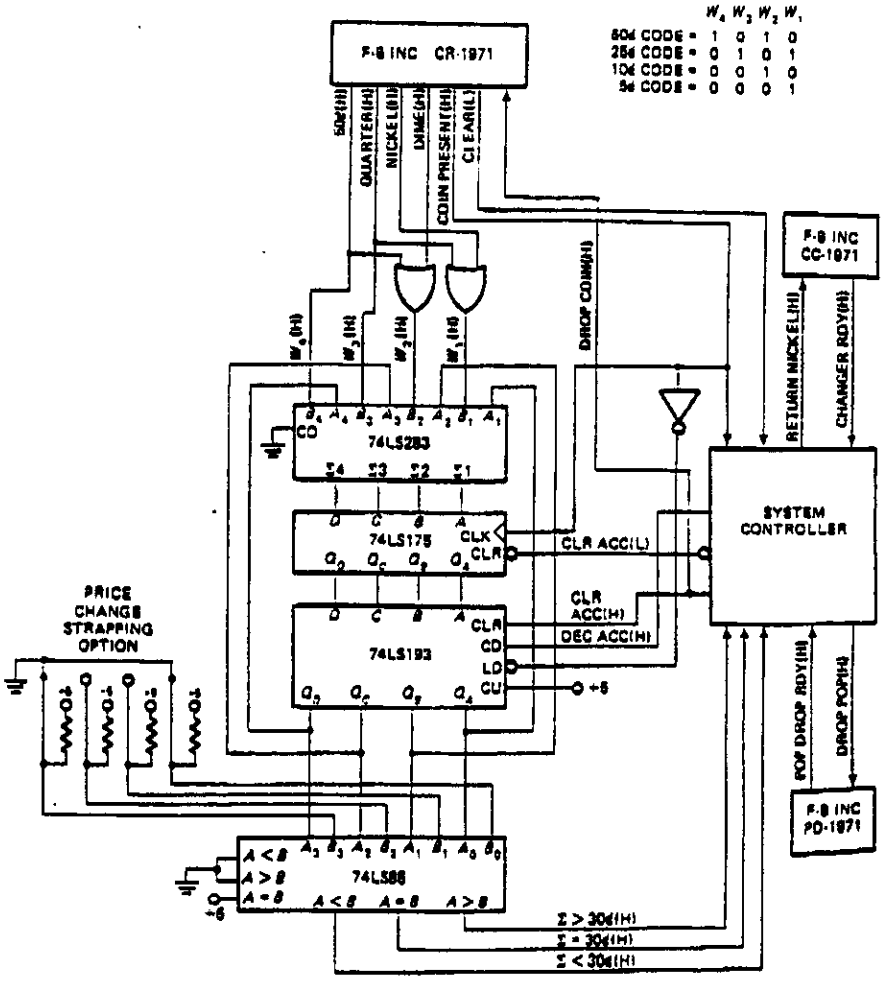


Fig. 7-41. The functional partition of the pop vending machine control system.

The seven-state MDS diagram for the system controller is shown in Figure 7-42. All branching conditions and outputs are clearly specified in close accordance with the specification set forth in the detailed flow diagram. The branching decisions based on asynchronous inputs are properly identified and are defined by the GO, NO-GO decision construct discussed in Section 7-10.1

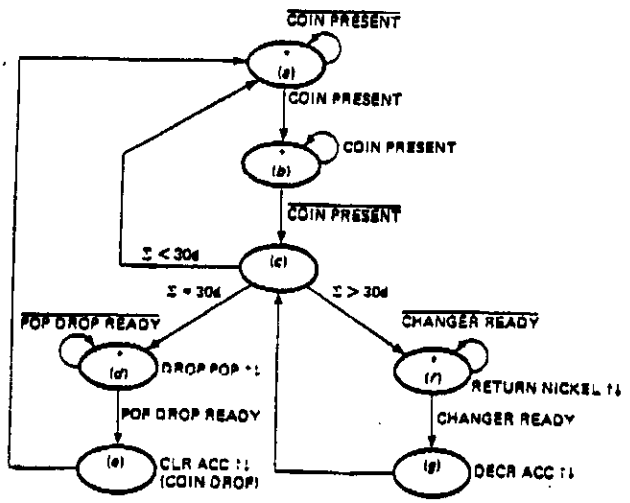


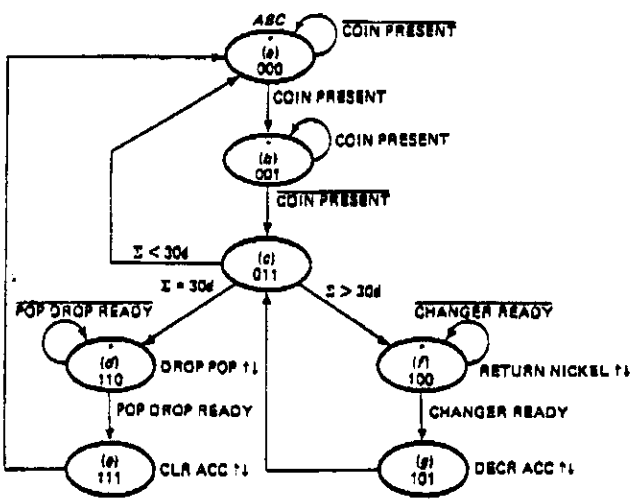
Fig. 7-42. The MDS diagram for the prototype pop machine system controller. Now that it is assumed CLR ACC ↑↓ causes the coins to drop.

7-11.5 THE HARDWARE IMPLEMENTATION OF THE NEXT STATE AND OUTPUT DECODER

Up to this point the major effort has been directed toward the system's three D's: Definition, Description, and Documentation. Now some of the practical aspects of an actual hardware development process are presented. The first step in this development process is to make a state assignment in accordance with the rules set forth earlier, paying strict attention to asynchronous branching (see Figure 7-43). A state map (regular K-map) is used to aid the state assignment process. Here the rules related to asynchronous branching variables are given the highest priority.

This priority calls for the following state assignment constraints:

a adjacent to *b*
b adjacent to *c*
d adjacent to *e*
f adjacent to *g*



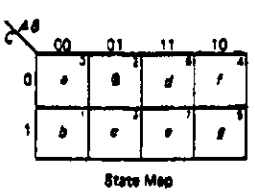


Fig. 7-43. The system controller MDS diagram for the prototype pop machine complete with a state assignment and state map.

The rest of the states are assigned with some reference to the simple rules set forth in Chapter 6 and the list in Table 7-1. However, no exhaustive attempt is made to optimize the state assignment.

RCL000076

TABLE 7-1 State assignment listing

Previous State	State in Question	Next State	Assignment Suggestions
a,c,e	a*	b	aADJb*; a,c,eADJ
a,b*	b*	b,c	bADJc*; aADJb*
b	c	a,d,f	a,d,fADJ; bADJc*
c,d	d*	e	dADJe*; cADJd
d	e	a	NR; dADJe*
c,f	f*	g	fADJg*; cADJf
f	g	c	NR; fADJg*

*→e must assignment; NR→NO Requirement

7-12 NEXT STATE DECODER MAPS

Once a state assignment is made, the next design step determines the first real commitment to any specific type of hardware. This example demonstrates the ease with which small sequential machines can be implemented at the SSI gate and Flip-Flop level, knowing well that other alternatives do exist. This ease is brought about in part by taking advantage of the inherent application of the VARIABLE-ENTERED MAP. Using VEM's bypasses the tedious detail of filling out a PRESENT/NEXT STATE TABLE by going *directly from the MDS diagram to the NEXT STATE maps*. The plotting of the VEM's is a simple process. However, it does require a *properly documented MDS diagram* and some ability to recognize, by inspection, the CONDITIONAL SET OPERATIONS called for in this MDS diagram. This process is described in the following paragraph.

Next State Map Plotting Process

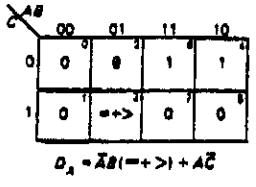
Using a MDS diagram with a state assignment, select a state, then examine the branching mnemonics to *all* NEXT STATES of the selected state; and enter into the STATE CELL of the NEXT STATE MAP the Boolean expression that defines all of the SET CONDITIONS (0→1, 1→1) that are required for the chosen state variable in that state. Thus these CONDITIONAL SET EXPRESSIONS become MAP ENTERED VARIABLES in the NEXT STATE MAPS.

This process is repeated for each STATE VARIABLE of each STATE until the NEXT STATE maps are completed. Remember, the NEXT STATE maps are the D-maps from which JK and T maps can be readily obtained. The described processes are clearly demonstrated in Figures 7-44 and 7-45. Pay particular attention to the entry in state cell c in the D_A map. Here you see:

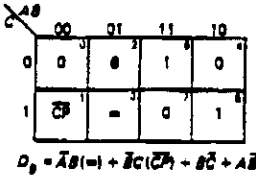
(⇒+>)

which indicates that the A Flip-Flop is set by the next clock edge if the machine is in: STATE c AND (=) OR (>) are ASSERTED. The rest of the MEV ENTRIES are obvious. However, it is recommended that a map be generated for each Flip-Flop and these maps be plotted in parallel, one state at a time, using the state map as a reference for the state cell location.

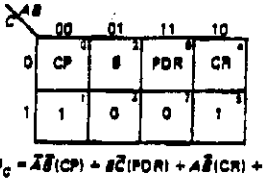
From Figures 7-44 and 7-45 notice that the JK does yield a circuit implementation requiring fewer gates (11 gates or four IC packages). Thus the new grand total for the system equals nine IC packages, some with spare gates, which really isn't too bad considering the capabilities of the overall system. However, the design for the OUTPUT DECODER must be completed.



$D_A = \bar{A}B(=+>) + A\bar{C}$



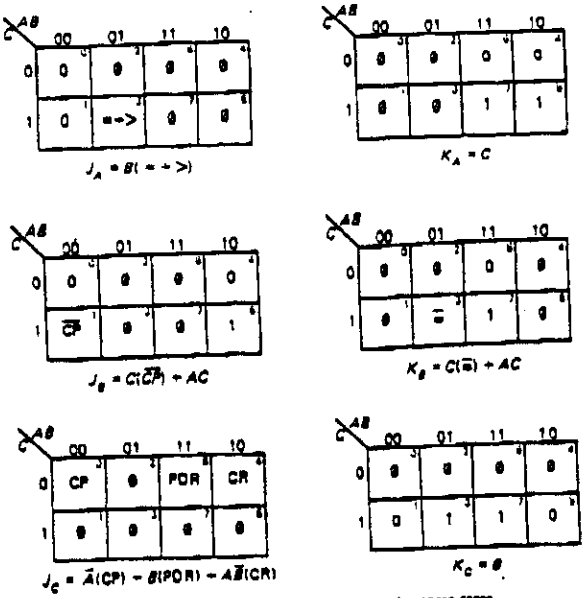
$D_B = \bar{A}B(=) + \bar{B}C(\bar{C}\bar{P}) + B\bar{C} + A\bar{B}C$



$D_C = \bar{A}\bar{B}(CP) + B\bar{C}(PDR) + A\bar{B}(CR) + \bar{B}C$

Legend: CP = COIN PRESENT
=+> = (Σ = 30d) + (Σ > 30d)
CR = CHANGER READY
PDR = POP DROP READY

Fig. 7-44. The NEXT STATE or D maps for the proto-type pop machine system controller.



Requires 11 gates, or four IC packages with spare gates.

Fig. 7-48. The NEXT STATE DECODER maps for a JK implementation.

7-13 THE OUTPUT DECODER

The design of the OUTPUT DECODER is like any other combination decoder design problem that can be summed up quickly by the following process:

> Make up a map for *each* output. Plot the map using MEV for the conditional variables, then simplify and implement.

However, many times maps serve no useful purpose. For example, in the present system controller there is no output that is a function of more than one (1) state. This information can be quickly gleaned from inspecting the MDS diagram. Therefore, in cases such as these, it is recommended that you make up a simple output list. This output list is nothing more than a list of every required output and

the states and conditions related to the generation of these outputs. For example, the MDS diagram in Figure 7-43 would have the following list:

$$\text{RETURN NICKEL} = A\bar{B}\bar{C}$$
$$\text{DECR ACC} = A\bar{B}C$$
$$\text{DROP POP} = AB\bar{C}$$
$$\text{CLR ACC} = ABC$$

From this expression list observe that none can be further simplified. Therefore, each requires a simple three-input gate (AND function) with the inputs connected to the PRESENT STATE VARIABLES (ABC). However, the ever present glitch problem exists because of our state assignment. The transition from 111→000 introduces the possibility of a glitch in three states. But the real question is which three states. Previous studies indicate that all six of the remaining states are vulnerable when the 111→000 transition is made. Therefore, some sort of glitch prevention must be introduced. In cases with a smaller number of bits changing (two in this case), the possible transitions can be traced in a state map for easy identification of possible glitch problems. However, the 111→000 transition overrides the need for any further glitch studies.

Since there are no outputs which must remain ASSERTED over several states, an OUTPUT HOLDING REGISTER would appear to be an overkill. However, DROP POP and RETURN NICKEL generated in states *D* and *F* must be held stable all during these states. Therefore, a holding register is probably called for. Therefore, four three-input NAND gates (74LS10) are chosen for the output decoder and a 74LS175 is chosen as the OUTPUT HOLDING REGISTER. For the completed schematic of the total system controller including the OUTPUT DECODER, see Figure 7-46. From this figure observe that the grand total IC package count is 14, all of which are standard or established 14 or 16 pin IC's (see parts list in Table 7-2).

TABLE 7-2 Parts List

Quantity	*Number*	*Description*
1	74LS283	Four-bit adder
1	74LS193	Hex up/down counter
2	74LS175	Quad D FIPO
2	74LS112	Dual edge-triggered JK Flip-Flop
1	74LS85	Four-bit comparator
2	74LS32	Quad of two-input OR
2	74LS10	Triple three-input NAND
1	74LS08	Quad of two-input AND
*1	7404	Hex of inverters
1	74LS00	Quad of two-input NAND
14		

*The standard 7404 was chosen because of the high drive requirement of the clock input on the 74LS112 (four std LS loads per input)

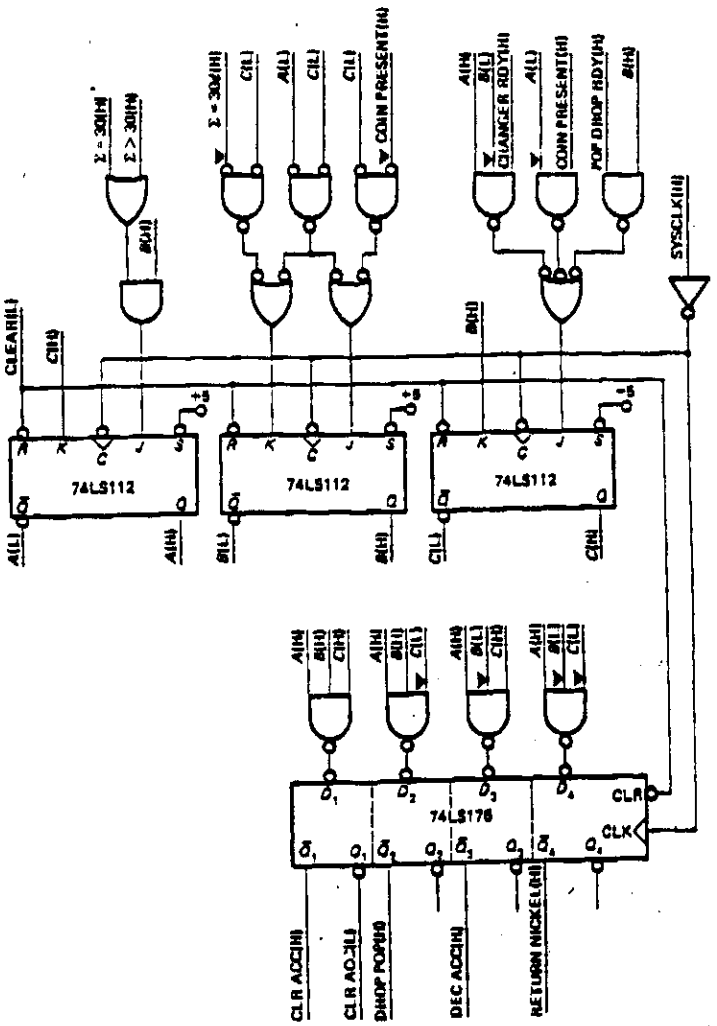


Fig. 7-46. The complete JK implementation of the system controller for the pop vending machine system.

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office

Address COMMISSIONER OF PATENTS AND TRADEMARKS
Washington D.C. 20231

SERIAL NUMBER	FILING DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NO.
07/143,601	01/13/88	TOMOHISA	3868-2

[illegible]
KELLY, SELTZER, PARK & GIBSON
POST OFFICE DRAWER 2410
CHARLOTTE, NC 28234

EXAMINER	
TRANS, V	
ART UNIT	PAPER NUMBER
234	3

DATE MAILED: 01/18/89

This is a communication from the examiner in charge of your application

COMMISSIONER OF PATENTS AND TRADEMARKS

☑ This application has been examined ☐ Responsive to communication filed on _____ ☐ This action is made final.

A shortened statutory period for response to this action is set to expire 3 month(s), _____ days from the date of this letter.
Failure to respond within the period for response will cause the application to become abandoned. 35 U.S.C. 133

Part I THE FOLLOWING ATTACHMENT(S) ARE PART OF THIS ACTION:

1. ☑ Notice of References Cited by Examiner, PTO-892.
2. ☑ Notice re Patent Drawing, PTO-948.
3. ☑ Notice of Art Cited by Applicant, PTO-1449
4. ☐ Notice of Informal Patent Application, Form PTO-152
5. ☐ Information on How to Effect Drawing Changes, PTO-1474
6. ☐ _____

Part II SUMMARY OF ACTION

1. ☑ Claims 1 - 30 are pending in the application.

   Of the above, claims _____ are withdrawn from consideration.

2. ☐ Claims _____ have been cancelled.

3. ☐ Claims _____ are allowed.

4. ☑ Claims 1 - 30 are rejected.

5. ☐ Claims _____ are objected to.

6. ☐ Claims _____ are subject to restriction or election requirement.

7. ☑ This application has been filed with informal drawings which are acceptable for examination purposes until such time as allowable subject matter is indicated.

8. ☐ Allowable subject matter having been indicated, formal drawings are required in response to this Office action.

9. ☐ The corrected or substitute drawings have been received on _____. These drawings are ☐ acceptable; ☐ not acceptable (see explanation).

10. ☐ The ☐ proposed drawing correction and/or the ☐ proposed additional or substitute sheet(s) of drawings, filed on _____, has (have) been ☐ approved by the examiner. ☐ disapproved by the examiner (see explanation).

11. ☐ The proposed drawing correction, filed _____, has been ☐ approved. ☐ disapproved (see explanation). However, the Patent and Trademark Office no longer makes drawing changes. It is now applicant's responsibility to ensure that the drawings are corrected. Corrections MUST be effected in accordance with the instructions set forth on the attached letter "INFORMATION ON HOW TO EFFECT DRAWING CHANGES", PTO-1474.

12. ☐ Acknowledgment is made of the claim for priority under 35 U.S.C. 119. The certified copy has ☐ been received ☐ not been received ☐ been filed in parent application, serial no. _____; filed on _____.

13. ☐ Since this application appears to be in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under Ex parte Quayle, 1935 C.D. 11; 453 O.G. 213.

14. ☐ Other

RCL000082

1. This application has been examined.

2. The following is a quotation of 35 U.S.C. 103 which forms the basis for all obviousness rejections set forth in this Office action:

> "A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made. Subject matter developed by another person, which qualifies as prior art only under subsection (f) and (g) of section 102 of this title, shall not preclude patentability under this section where the subject matter and the claimed invention were, at the time the invention was made, owned by the same person or subject to an obligation of assignment to the same person."

This application currently names joint inventors. In considering patentability of the claims under 35 U.S.C. 103, the examiner presumes that the subject matter of the various claims was commonly owned at the time any inventions covered therein were made absent any evidence to the contrary. Applicant is advised of the obligation under 37 CFR 1.56 to point out the inventor and invention dates of each claim that was not commonly owned at the time a later invention was made in order for the examiner to consider the applicability of potential 35 U.S.C. 102(f) or (g) prior art under 35 U.S.C. 103.

3. Claims 1-30 are rejected under 35 U.S.C. 103 as being unpatentable over Darringer et al (U.S. Patent No. 4,703,435) or Darringer et al in view of Nash et al (EDA84 - A Front End Graphic Interface to the First Silicon Compiler).

Darringer et al disclose a method and system for an automatic logic design that logic is synthesized from a flow-chart level description. It is clear to one skilled in art that a series of action and condition blocks interconnected to form a flow-chart representing a functional specification for the integrated circuit (logic circuit), and it is known in the art of automatic layout that the automatic layout method and system have classically relied on a library of circuit components or cells in the forms of

mask geometric defining logic gates to place and interconnet the cells on a substracte to form an integrated circuit.

As per claims 3 and 15, although Darringer et al do not specifically disclose a flow-chart editor means for creating a flow-chart having elements representing actions and conditions operating by user, Nash et al disclose a graphic editor driven by the knowledge base to allow user to creat a schematic or flow-chart on the graphic screen and said schematic or flow-chart to be compiled by LANGUAGE COMPILER. Since both Nash et al and Darringer et al are directed to a system and method for compiling an flow-chart level description through language compiler to generat a physical layout circuit, it would have been obvious to one skilled in the art to employ Nash teaching into Darringer's invention to produce the claimed invention.

5. The additional cited references are considered as art being relevant to this case. Applicant is requested to consider them fully when responding to this office action.

6. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Vincent Trans whose telephone number is (703) 557 - 8005. Any inquiry of a general nature or relating to the status of this application should be directed to the Group receptionist whose telephone number is (703) 557-2878.

V. Trans

PARSHOTAM S. LALL
SUPERVISORY PATENT EXAMINER
ART UNIT 234

TO SEPARATE, HOLD TOP AND BOTTOM EDGES, SNAP-APART AND DISCARD CARBON

1 OF 2

FORM PTO 892 (REV. 3-78) U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

NOTICE OF REFERENCES CITED

SERIAL NO.	GROUP ART UNIT	ATTACHMENT TO PAPER NUMBER
07/143,821	234	3

APPLICANT(S): Kobayshi et al.

U.S. PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
	A	4,703,435	10/27/87	Darringer et al.	364	488	7/16/84
	B	4,700,317	10/13/87	Watanabe et al.	364	521	11/28/84
	C	4,656,603	4/7/87	Dunn	364	488	3/1/84
	D	4,651,284	3/17/87	Watanabe et al.	364	491	7/26/85
	E	4,638,442	1/20/87	Bryant et al.	364	489	11/6/84
	F	4,635,208	1/6/87	Coleby et al.	364	491	1/18/85
	G						
	H						
	I						
	J						
	K						

FOREIGN PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	COUNTRY	NAME	CLASS	SUB-CLASS	PERTINENT SHTS. DWG	PERTINENT PP. SPEC.
	L	1,445,914	8/11/76	G.B.	Colton et al.	364	490		
	M								
	N								
	O								
	P								
	Q								

OTHER REFERENCES (Including Author, Title, Date, Pertinent Pages, Etc.)

R "Verifying Compiled Silicon" by E.K. Cheng, VLSI Design, Oct. 1984, pp. 1 ~ 4.

S "CAD System for IC Design" by M.E. Daniel et al., IEEE Trans. on Computer-Aided Design of Integrated Circuits & Systems, Vol. CAD-1, No. 1, January 1982, pp. 2 ~ 12.

T "An Overview of Logic Synthesis System" by L. Trevillyan, 24th ACM/IEEE Design Automation Conference, 1978, pp 166 - 172.

U

14

EXAMINER	DATE	
V. TRANS	11/18/88	RCL000085

* A copy of this reference is not being furnished with this office action.
(See Manual of Patent Examining Procedure, section 707.05 (a).)

1445914 COMPLETE SPECIFICATION

6 SHEETS *This drawing is a reproduction of the Original on a reduced scale*

Sheet 1

FIG. 1

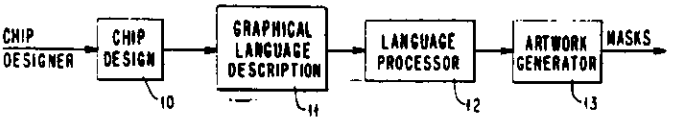
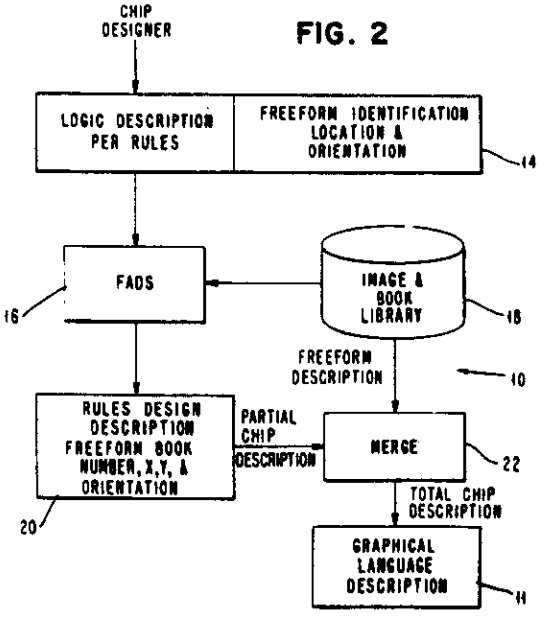


FIG. 2

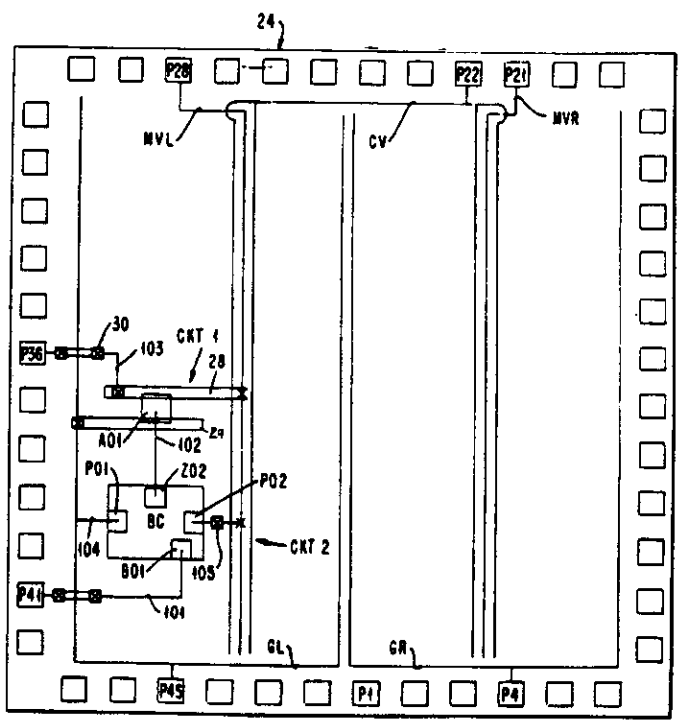


RCL000086

IBMC E1880W/16 =GB 1445-914
Automatic designing of integrated ccts. - is based on determination of desired masks forming program controlled system
IBM CORP 27.09.73-US-401303
R46 *R24 R27 (11.08.76) *DT2442-850 *G06f-15/20
The apparatus is designed for producing graphical data descriptive of an integrated circuit design. The apparatus includes a store for graphical descriptions of freeform designs and a store for topological data and electrical characteristics of rules restricted designs. Information is entered denoting the location and relationship of a freeform design and a rules restricted design. A graphical description of this rules restricted design is generated and the generated graphical description of this rules restricted design is merged with the graphical description of the stored freeform design to form the graphical data descriptive of an integrated design. 30.8.74 as 037953. (13 pp).

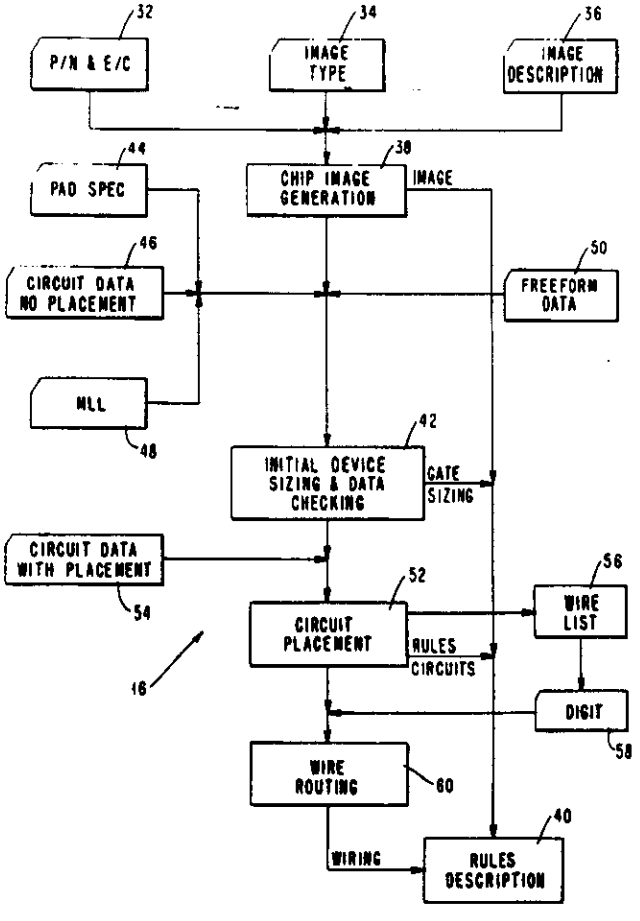
1445914 COMPLETE SPECIFICATION
6 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 2

FIG. 3



1445914 COMPLETE SPECIFICATION
6 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 3

FIG. 4



RCL000088

1445914 COMPLETE SPECIFICATION
6 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 4

FIG. 5a

46 (54)

CIRCUIT NUMBER	BOOK NUMBER	COLUMN	ROW	ORDER	ORIENTATION	LOAD DEVICE	LOAD WIDTH	FUNCTION	SUB-FUNCTION	SPECIFICATION	SPECIFIED LST	DELAY LST
1	1	(1)	(2)	(1)	(1)							

CIRCUIT DATA

FIG. 5b

50

CIRCUIT NUMBER	BOOK NUMBER	X LOCATION	Y LOCATION	ORIENTATION
2	401	10	50	1

FREEFORM DATA

FIG. 5c

48

NET NUMBER	CIRCUIT NUMBER	LST ID	CIRCUIT	LST ID	CIRCUIT	LST ID
101	2	B01	-1	P41		
102	2	Z02	1	A01		
103	1	Z01	-101	P36		

MASTER LOGIC LIST (MLL)

1445914 COMPLETE SPECIFICATION
6 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 5

FIG. 5d

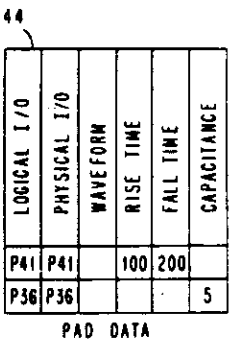


FIG. 5e

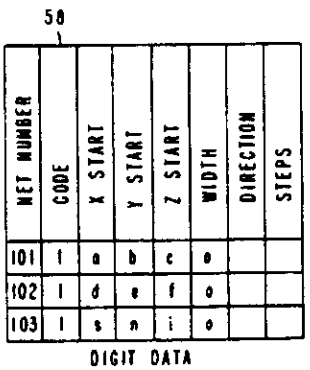
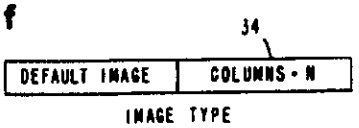
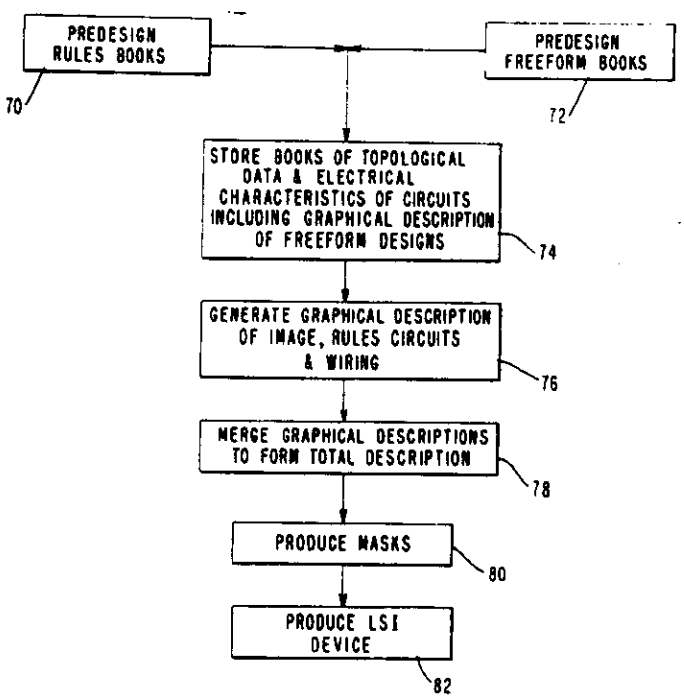


FIG. 5f



1445914 COMPLETE SPECIFICATION
6 SHEETS *This drawing is a reproduction of the Original on a reduced scale*
Sheet 6

FIG. 6



235/151

AUG 1976

PATENT SPECIFICATION

(11) **1 445 914**

1 445 914

(21) Application No. 37953/74 (22) Filed 30 Aug. 1974
(31) Convention Application No. 401 303
(32) Filed 27 Sept. 1973 in
(33) United States of America (US)
(44) Complete Specification published 11 Aug. 1976
(51) INT CL² G06F 15/20
(52) Index at acceptance G4A 5B 9X U

(72) Inventors WILLIAM FRANCIS COLTON, BELA GOGOS, WILLIAM ROSENBLUTH and DOUGLAS HUBERT RUTHERFORD

(19)

GREAT BRITAIN
GROUP 226
CLASS 235
RECORDED

(54) IMPROVEMENTS RELATING TO APPARATUS FOR PRODUCING GRAPHICAL DATA DESCRIPTIVE OF AN INTEGRATED CIRCUIT DESIGN

(71) We, INTERNATIONAL BUSINESS MACHINES CORPORATION, a Corporation organized and existing under the laws of the State of New York in the United States of America, of Armonk, New York 10504, United States of America, do hereby declare the invention, for which we pray that a patent may be granted to us, and the method by which it is to be performed, to be particularly described in and by the following statement:—

This invention relates to apparatus for producing graphical data descriptive of integrated circuit semiconductor chip design. It is concerned in improving the design of semiconductor chips by means of design automation techniques to produce chip design data that may be used to control machines which make photolithographic masks used to manufacture the semiconductor chips.

The use of design automation for making large scale integrated (LSI) chips and the advantages thereof are disclosed in "Computer", Vol. 5, No. 3 pages 18—52, published by the IEEE Computer Society and in the complete Specifications of our copending Applications for Letters Patent Nos. 25812/73 and 58461/72 (Serial Nos. 1 414 018 and 1 405 508 respectively. LSI devices can be generally classed as "non-random" and "random" types. A "non-random" LSI device is characterized by an array of duplicate circuits such as might be used in a memory device or a read-only store (ROS). A "random" LSI device is characterized by the lack of any repetitive uniformity of circuits on a chip. The circuits, and functions performed thereby, may vary on a single chip. An example would be a chip containing combinational and sequential logic for use in the control section of a computer. Quite obviously, designing random LSI chips is a difficult task and it is within such an art that design automation techniques are particularly advantageous and it is the general area which the present invention improves.

Before the invention is explained in detail a general design process will be described. A prerequisite to making a given device is the existence of a given technology in which the process of making the device is established and the electrical characteristics of the circuits and packaging are known. Design criteria or rules for the technology are established. With the knowledge of such rules, a chip designer performs the complex task of translating logical functions to be performed in a system into actual physical LSI devices that perform the functions. To do this, the designer interacts with and is aided by a design automation (DA) system. The theory for use of any DA system is that there are many operations which are subject to algorithmic analysis and control which can be best done by a computer while there are other functions that require human skill to be performed effectively or best. In designing random type LSI chips, a chip designer is "best" at deciding what functions, and circuits to perform such functions, are to be placed on a given chip and where elements should be generally placed. A chip designer can place many relatively small diverse functions on a given chip or he may break up a function and place it on different chips. The chip designer produces an output that is partially descriptive of a chip. This output is fed as an input to a DA system which performs its functions to provide an output that is used to control the physical process of making the LSI device. This would entail making photolithographic masks.

In the design process there are two mutually exclusive techniques, "free form" design and "rules driven" design. The

former technique is unrestricted and characterized by efficient silicon or chip utilization but has a long design cycle (e.g., three to six months) and requires manual checking. The latter technique is restricted and characterized by a fast design cycle (e.g., two to four weeks) and automatic checking but may produce inefficient silicon utilization.

A specific example of the restricted "rules driven" technique used in connection with FET (field effect transistor) LSI devices is disclosed in the aforementioned complete Specification of our Application for Letters Patent No. 58461/72 (Serial No. 1 405 508). In accordance with such "rules driven" technique, the metallization areas, diffusion areas, power and signal busses, I/O pads and FET devices are laid out on an orthogonal grid pattern in accordance with relatively rigid or restricted rules, whereby the maximum use can be made of the DA system to provide a fast design cycle.

An object of the present invention is to provide an apparatus that combines both "free form" and "rules driven" techniques.

According to the invention, apparatus for producing graphical data descriptive of an integrated circuit design comprises first means adapted to store graphical descriptions of freeform designs, second means adapted to store topological data and electrical characteristics of rules restricted designs, means adapted to enter information denoting the location and relationship of a freeform design to a rules restricted design, means adapted to generate a graphical description of said rules restricted design, and means adapted to merge the generated graphical description of said rules restricted design with the graphical description of said freeform design stored in said first means to form said graphical data descriptive of an integrated circuit design.

In the embodiment to be described below, a library of predesigned graphical descriptions that describe circuits designed by the free form technique is stored within the design automation system. A chip designer would design the layout of a chip including both "free form" and "rules driven" portions. An input is fed to the DA system which specifies the rules driven design in the established manner and specifies the free form design by defining the library description thereof, and the relative placement of the design on the chip. The system in response to such input generates a graphical description of the rules driven design without doing any automatic checking of the internal connections etc of the free form design. The generated description or topology is then merged with the free form description or topology from the library to form a complete chip description from which is derived data for making photolithographic masks.

The invention will now be particularly described, by way of example, with reference to the accompanying drawings, in which:—

Figure 1 is a general flowchart showing the overall design process,

Figure 2 is a general flowchart showing the overall chip design process of Figure 1 in more detail,

Figure 3 is a diagrammatic layout of a chip design providing an example useful in understanding the invention,

Figure 4 is a more detailed flowchart,

Figures 5a to f illustrate various descriptive data related to the example shown in Figure 3, and

Figure 6 is a general flowchart summarizing the steps in the design process.

Referring now to Figure 1, in accordance with the overall or general process, a chip designer provides an input to a chip design system 10, the details of which will be discussed hereafter. The input of the chip designer incorporates both free form data for achieving efficient silicon utilization and incorporates rules driven data for taking maximum advantage of the speed and efficiency of the chip design system 10. The output of chip design step 10 is a graphical language description 11 of the eventual chip design. This description is fed as an input to a language processor 12 that provides an output for controlling the operation of an artwork generator 13 used to produce the masks for making the actual physical LSI chip. This overall process is disclosed in the aforementioned complete Specification of our Application for Letters Patent 58461/72 (Serial No. 1 405 508). A difference is that the chip design system 10 is modified and the addition of the freeform input by the chip designer.

Referring now to Figure 2, at the start of the process, the chip designer would provide input information 14 to the FET automated design system (FADS) 16. To do this, the designer, with knowledge of the functions to be incorporated into a chip, selects the desired circuits and devices from those available to him within the given technology, including those designs that are rules driven and those that represent a freeform design. The chip designer would ordinarily work with a graphical layout of the chip, such as illustrated in FIG. 3, which includes an image of the general grid structure including the power busses and I/O pads. Upon this layout, the designer would then

place the respective circuits. Stored within the system is a book library 18 which is a file of technology data describing the electrical characteristics of circuits, topology thereof and the images. The exact or specific information would be dependent upon the functions to be performed by the system which can vary from system to system. The information in the library may be divided into three types of books or collections of data. Types 1 and 2 respectively represent circuits that are specified by formulae so that the system calculates devices, shapes and locations, or may specify standard circuits conforming to the given rules. The third book type would be those providing the freeform topology.

Let us consider now in a little more detail the process used by the chip designer. The chip designer might first layout in a block diagram form the logical functions to be included within the chip as, for example, through the use of And and Or blocks. This would be the logic representations of the functions to be eventually created and performed within an actual chip. Next, the chip designer would select a chip image such as that shown in FIG. 3 and beging a layout of the various circuits to be included. He would first determine what devices or circuits are necessary to perform the logic functions and then would begin relative placement of these devices on the chip image. In the illustrated example, the chip image has 48 terminals and a bus network that divides the chip into four active areas in the form of columns or bays. A given circuit would be generally located in one of these bays. Also, the designer would decide which relative row of a given bay the circuit should go in, the relative row being the degree of vertical stacking of circuits within a given bay. This process is known as a placement of the devices on the image. Later, FADS 16 will determine exact placement. The relative placement can be done by assigning XY coordinates to the chip image which XY coordinate might for example, by convention, be considered to be the location of the lower lefthand corner of any given area of diffusion, contact, or metallization. Through use of the chips designer's skill, the various functions would be implemented within circuits and would attempt to provide maximum efficient use of the active area of a chip. The input information 14 is then generally divided into two aspects. The first is that connected with the description of the logic in accordance with the rules of the system and the second would be the description identifying the freeform device and its location and orientation.

FADS 16 is constructed in accordance with standard DA techniques and comprises a series of DA programs that are loaded in and executed on a general purpose computer such as a Model 158 of IBM (Registered Trade Mark) System/370. The language of the program can be any general programming language such as PL/I which act upon the data provided by the graphical programming language to produce the design of the chip. In general, FADS 16 would, using input data 14 and information from library 18, construct an image of the circuits on the chip, except for the freeform, and convert any relative placements of the input into exact placements. The freeform itself would be treated as an area which has signal and power terminals to be accounted for, but the FADS system would do no internal checking of the freeform design to see that it worked. In other words, it would be the responsibility of the designer to make certain that the freeform design operated correctly. When FADS 16 has completed its processing, the result is output 20 which contains a partial description of the chip design for the rules design portion thereof, along with an identification of the freeforms to be incorporated. This partial chip description is then fed through a merge process 22.

Freeform identification of output 20 is used to obtain from library 18 the freeform description which in the merge process 22 is combined with the partial description from the output 20 to form a total chip description or graphical language description 11.

In the beginning of the design process, the chip designer would select a chip image layout on which to begin placing the various circuits and devices. Such a layout is shown in FIG. 3 which schematically illustrates a partially completed LSI chip 24. Such a device is constructed in accordance with the manner pointed out in the aforementioned Specification and generally comprises a silicon wafer having diffusions therein, such as 28, described in more detail below, forming part of the conductive pattern on the wafer and acting as part of the circuit devices. Two layers of thick and thin oxides are placed on top of the wafer and any metallization is done on top of the oxide layers. Contacts 30 provide vertical electrical connection between diffusion and metallization zones. It is to be understood that in a given technology system there might be several different images or basic images available to a designer that vary according to the number of I/O pads and chip size. In the illustrated chip 24, there are forty-eight I/O pads P1—P48 located around the periphery,

each pad being rectangular in shape and comprising metallization and diffusion. These pads are designed to be connected to external terminals for providing signals and power to chip 24. Thus, pad P1 is designed to power the substrate and it would include a contact leading from the upper metallization layer down to a diffusion area on the substrate. Pads P45 and P4 are designed to be connected to an external ground connection and are also connected to metallization busses GL and GR for providing a ground network on the left and right sides of the chip. Pads P21 and P28 are designed to be connected to voltage sources to provide main voltage busses MVL and MVR for the left and right sides of chip 24. Pad P22 is designed to be connected to a source of control voltage and is also connected to a control bus CV whereby a voltage applied to this bus will allow the circuits on the chip to be operated. Bus CV includes bifurcated arms extending parallel to busses MVL and MVR and having legs lying along each side of each bus in the manner pointed out in the above applications. The various vertical busses in the specific examples shown in FIG. 3 divide the active area of the chip into four columns in which the circuits and wiring are to be placed.

To illustrate the invention, FIG. 3 shows a rules design circuit CKT1 and a freeform or predesigned circuit CKT2. These circuits are shown merely for the purpose of illustrating the invention. In the case of the design of an actual chip, there would be many more circuits included to utilize the active area of the chip as effectively as possible. Circuit CKT1 is an FET device used to invert an output signal appearing from CKT2. CKT2 is a binary counter of conventional logic instruction having an input terminal BO1, and an output terminal ZO2. The function of the counter is to produce an output signal on the output terminal ZO2 upon receiving four input pulses or signals at input terminal BO1. Input terminal BO1 is connected by a network 101 to I/O pad P41. Output terminal ZO2 is connected by network 102 to the metal cap AO1 of CKT1. Diffusion rails 28 and 29 underlie cap AO1 but are separated therefrom by oxide layers. Rail 28 is connected via network 103 to I/O pad P36. The binary counter CKT2 is also provided with two power connectors PO1 and PO2 respectively connected to busses GL and MVL.

The flowchart of FIG. 4 is an expansion of FADS 16, Figure 2. Once the chip designer has completed the layout of the entire chip, the information can be translated into data that serves as an input to the FADS system. In general, there are four steps or operations in this system, the result of which is to produce the rules description 40 described above. Each step has an input from external information and from the output of a preceding step and each step provides partial information or generates a portion of the rules description 40, as pointed out in more detail hereafter. Within this diagram, while the various forms of external data input are shown schematically as coming from a punched card deck, it is to be understood that the input can be by any other standard input method associated with a data processing system. In general, as indicated previously, FADS 16 is designed primarily to handle the rules restricted design but is modified to recognize the existence of freeform designs.

The first function is chip image generation 38, the purpose of which is to allow the designer to select and, modify if he so desires, a standard image available to the system. The output of this step would include a description of such an image that would eventually be incorporated into the description 40. To obtain this output, three inputs are provided. The first input 32 identifies any part number and engineering change level associated with a given LSI chip 24. A second input 34 defines the basic image type to be used by the designer. The input can be information similar to that shown in FIG. 5f which includes a first field identifying the image and a second field identifying any particular columns where such a choice is available. Within this particular example, the image shown in FIG. 3 is a "default image" so designated in FIG. 5f, having four columns of active areas. A third type of input 36 provides an image description of the manner in which the basic image is to be modified for use by the designer if he so chooses. Such modifications might come about by bending the power busses. In the example, no modification is necessary.

The second operation 42 is to perform the functions of initial device sizing and data checking. The designer defines the logic to be implemented and the performance required from the logic. The system then calculates the device sizes required for each circuit to meet its specified performance. In general, the logic is described in terms of nets or grouping of circuit LSTs (logic service terminals) and I/O pads which must be connected to implement the logic function. Performance or speed requirements are indicated to the system by specifying the input waveform and/or capacitive loading present at the I/O pads and the performance of each circuit. During this operation, the system validates the designer's input and may perform a

number of checks including the following: each circuit has been equated to a book type: master logic list (MLL) is consistent with circuit data: fan-in to a circuit; number of outputs dotted; total fan-in to dotted circuits; load device has been specified for one circuit in a dotted net. It should be obvious that many of these checks are equated to only the rules restricted design and the checks are merely to make certain that the design conforms at least to certain ones of the rules checked for in this operation.

Operation 42, in addition to receiving as an input the output from the preceding step 38 receives a first input 44 called the pad spec or specification shown in detail in FIG. 5d. With reference to such Figure, the pad spec includes two fields for correlating the logical I/O numbers used by a designer to distinguish an I/O pad with an actual physical I/O pad number. The physical I/O number is optional and if the designer does not specify a particular one, the system will determine or specify one. In the particular example, the two physical pads P41 and P36 are also considered to be the same logical I/O pad numbers. The waveform field might be used to specify any special characteristics of a waveform. Where the field is not used, the pad is not a primary driver and receiver. The rise time specifies for example in nanoseconds the time of the rising input waveform where the pad is a primary input or has an off chip dot connection. In the example, it is assumed that the waveform associated with pad 41 has a rise time of 100 nanoseconds and fall time of 200 nanoseconds. The remaining field is the capacitance load expressed for example in picofarads, as seen by the circuit which is driving the I/O pad and should be specified if the I/O pad is primary output or has an off chip dot.

Another input to step 42 is MLL 48 shown in FIG. 5c. In general, the MLL describes the set of connection required to implement the desired logic. Each connection or net must be assigned a unique positive integer by the designer. The net is composed of two types of connections, circuit LSTs and I/O pads. The first field is the net number which in the particular example involves three nets numbered 101, 102 and 103. Next will come a series of paired fields identifying a circuit number and LST ID, the number of these fields being repeated for as many as are associated with a given net. A negative circuit number indicates an I/O pad, a positive number indicates an LST. For the specific example, shown in FIG. 3, FIG. 5c includes the specific or exemplary information shown.

The next input 46 to step 42 is the circuit data information as in Figure 5A, used to uniquely identify circuit blocks on the logic diagram and attach physical attributes to it and to specify circuit specifications or data in accordance with any terms defined within the associated book library. This data is initially supplied to step 42 without any placement information and is later modified to include the placement information as an input to a succeeding step as described below. The first column specifies the circuit number which in the example is 1. The next field identifies the book number which equates the circuit number to a logic function and topology stored in a book library 18. The next three columns or fields of column row and order are used to identify the relative placement of the circuit on the chip. In the example, the chip has four columns with the left column being numbered 1. The row refers to the relative vertical stacking within a given column where circuit 2 occupies the first row and circuit 1 occupies the second row. This is really relative and spaces may be skipped in between. The system will eventually determine the exact placement. Order refers to the number of circuits within a given row in a column and there may be more than one device although in the example, only one device A01 is shown and this would have an order of 1. The next field orientation refers to the basic orientation as stored in the book library and whether it is to be mirror imaged about the X or Y axes. The remaining fields relate to certain circuit specifications or data which can be filled in or left blank in which case the default parameters would be used.

The remaining input to step 42 is freeform data 50 shown in detail in FIG. 5b. The first field identifies the circuit number. The next field identifies the book number which in this example is an arbitrary number 401 that would identify the associated topological and electrical characteristic information within book library 18. The X location and Y location would refer to the relative coordinates for the lower lefthand corner of the outline of circuit 2. The active area of the chip may be arbitrarily divided up into grid units each grid unit representing for example 0.025 microinches. The orientation of 1 indicates that it is to use the basic orientation within the book with no mirror imaging around the X or Y axes. This information is then used by the system to more or less outline or create a hole within which the processing relative to the other circuits proceed without any attempt to be made to check what is within the outline. The specific topology of the circuit is obtained

from the library as well as the locations of the terminals thereon so that the specification of the location of the corner of the circuit along with its orientation, is used to specify the location and also indirectly the geometry of the circuit.

The third operation 52 is one of circuit placement and this requires the designer to assign logic circuits to positions on the chip. The assignment is done in terms of relative coordinates which are automatically transformed to absolute coordinates by the system, the data or assignment being included as placement data 54 associated with the circuit data. This information is described previously relative to FIG. 5a wherein in step 54, the information enclosed in parenthesis is added.

The output from the circuit placement operation 52 includes a wire list 56 that indicates the coordinates for each LST which must be wired to implement the logic. It could also include any information for modifying the device size and delay that's based upon any revised loading or electrical parameters derived from the placement data. The chip designer would then utilize this wire list to produce an input 58 containing digit data information shown in FIG. 5e. the first field contains the net number corresponding to the same net number in MLL 48. The second field can be used to distinguish a code between whether the specifications are digitized on wiring channels or in grid units. The next three fields define the X, Y and Z coordinates for the starting point of the first line segment, the Z coordinate corresponding to the chip layer or mask level in which the wiring starts. The width field can be used by putting a zero therein to indicate that a standard width of metallization or diffusion is to be used or it can specify the actual width. The remaining fields, and there can be more than one dependent upon how many changes there are in the line segment, can be used to first define the direction of movement in either positive or negative X or Y direction from the starting point or a vertical direction and the steps field will specify the number of grid units or wiring channels along which the line segment will extend. If the direction is specified as indicating a change between layers, the steps field can then be used to specify a type of contact for accomplishing the change. The X and Y coordinates define the location of the wire or LST relative to the origin of the chip.

The final step 60 of the process is a wire routing operation that utilizes the digit data and output from step 52 to route wires to interconnect all the circuit LSTs and I/O pads. The wire routing step checks to see that the logical circuit LSTs coded in the MLL are matched against gates placed in the digitization. It also checks for net continuity, intersection nd critical spacing. The final output of the wiring routing step 60 is placed in 40 and completes the description of the chip except for the specific information stored in the book library describing the internals of the freeform design. With the design completed, various other tests such as delay calculations, may be performed to aid the designer and help ensure the accuracy of the completed design.

Referring now to FIG. 6, the overall steps in the design process are summarized in flowchart form. Initially, step 70 involves the predesign of those circuits associated with the rules restricted design. Step 72 involves predesigning the freeform designs and organizing the data into books. As indicated previously, the system will operate on the rules books 70 to perform as many DA functions relative thereto as possible whereas the number of functions performed on the freeform books 72 is restricted in that no checks are made by the automated design system relative to any of the internal workings of the devices. These predesigns are converted into data acceptable to the data processing system and, in step 74, they are stored within the system in books containing the topological data and electrical characteristics of the circuits and graphical descriptions of the freeform designs. Thereafter, step 76 generates the graphical description of the image, rules circuits and wiring, such graphical description of the freeform design being already stored in book form. Next, step 78, the description from step 76 is merged with the description of the freeform design as obtained from the library, the merger producing the graphical description of the total device. Thereafter, the appropriate photolithographic masks are made in step 80 for use in the step 82 to actually produce an LSI device.

WHAT WE CLAIM IS:—

1. Apparatus for producing graphical data descriptive of an integrated circuit design comprising first means adapted to store graphical descriptions of freeform designs, second means adapted to store topological data and electrical characteristics of rules restricted designs, means adapted to enter information denoting the location and relationship of a freeform design to a rules restricted design, means adapted to generate a graphical description of said rules restricted design, and means adapted to merge the generated graphical description of said rules restricted design with the graphical description of said freeform design stored in said first means

to form said graphical data descriptive of an integrated circuit design.

2. Apparatus as claimed in claim 1, wherein said entering means comprises means adapted to specify the relative location on said freeform design of signal and power terminals, and means adapted to specify conductive networks interconnecting said terminals with said rules restricted design.

3. Apparatus as claimed in claim 2 comprising means adapted to store topological information providing relative orientations of any design in the form of mirror images about Cartesian axes, and means adapted to specify the desired orientation of said freeform design.

4. Apparatus as claimed in claim 1, comprising means adapted to specify the location of said freeform design relative to an origin on said chip of a grid unit locating coordinates.

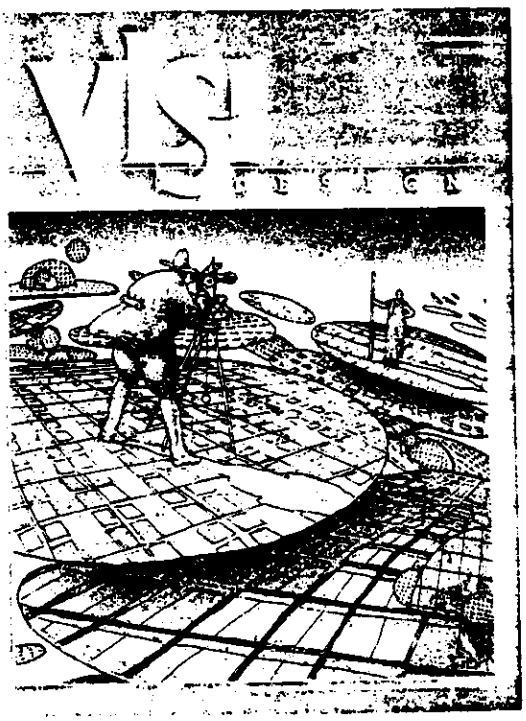
5. Apparatus as claimed in any preceding claim, comprising means adapted to store a graphical description of at least one image for said integrated circuit design defining the topology of power busses and input-output pads.

6. Apparatus for producing graphical data descriptive of an integrated circuit design, substantially as herein described with reference to the accompanying drawings.

JOHN BLAKE,
Chartered Patent Agent,
Agent for the Applicants.

Printed for Her Majesty's Stationery Office by the Courier Press, Leamington Spa, 1976.
Published by the Patent Office, 25 Southampton Buildings, London, WC2A 1AY, from which copies may be obtained.

3642/488



Cover

Surveying the expanding universe of application-specific ICs may be unsettling. A stable basis for comparison is hard to find in a shifting landscape. Cover illustration by Gary Lund, Venice, CA.

Copyright© 1984 by CMP Publications, Inc., 600 Community Drive, Manhasset, N.Y. 11030.
Reprinted with permission from VLSI Design.

Verifying Compiled Silicon

Edmund K. Cheng, Silicon Compilers, Inc., Los Gatos, CA

As the complexity of VLSI chips increases, the intuitions of design engineers fail more often. The large number of components and their complex relationships conspire to obscure the detailed consequences of each design decision. At the same time, the cost of making a mistake has escalated sharply, even in prototyping. To minimize the likelihood of costly errors, designers increasingly depend on computer tools to synthesize and verify designs before committing them to the costly fabrication process.

Verification tools fall into two categories: static and dynamic. Static tools check a design against a defined set of rules and flag any violations. Common static tools include design-rule checking (DRC), electrical-rule checking (ERC), and timing analysis. Dynamic tools (such as circuit, logic, and register-transfer-level simulators) imitate the behavior of a design, thus allowing the engineer to test and understand a circuit or system in a variety of environments. Conventional IC design requires three manual translations in traversing the four design steps from architecture to logic to transistor circuit to layout. As a result, most of the design verification efforts focus on establishing the equivalence of the different design representations.

Silicon Compilers, Inc. recently introduced a system (Genesil Silicon Development System) that shifts the process of designing VLSI chips towards the architectural level and away from the classical IC-design considerations of logic and layout topology. Such a shift implies changes in verification methods as well as design methods. This article describes VLSI design verification in the context of this silicon compiler, namely, functional simulation and timing analysis. We treat the two functions separately, describe how they work, and contrast them with conventional verification.

Silicon Compilation

Just as software compilation synthesizes machine-language code from programs written in a high-level language, the Genesil Silicon Development System synthesizes relevant views of an IC from high-level architectural descriptions. These views are derived from descriptions in terms of structures "known" to the compiler, such as PLAs, ROMs, RAMs, ALUs, registers, and data paths. These structures can be described functionally, in quantity, or in terms of mathematical, logical, or state equations. The design system then synthesizes views of the layout (IC topology), the function (simulation model), and the performance (timing model) of the specified architecture.

The physical circuitry is algorithmically generated using layout/circuit primitives that have been verified previously (using conventional verification tools). Layout design of the "known" structures in this system can therefore be characterized as "correct-by-construction." Architectures may be specified hierarchically and the layouts may be constructed hierarchically (wiring blocks into modules and modules into chips). Assembling modules of compiled blocks requires manual placement of block outlines. The system then automatically routes interconnections.

Equivalence Check	Traditional Method	Silicon Compilation
Wish to Architecture	RTL Simulation	Functional Simulation/
Architecture to Logic	Logic Simulation	(Implicit)
Logic to Circuit	Switch Simulation	(Implicit)
Circuit to Layout	Extraction	(Implicit)
Layout (Absolute)	DRC	(Implicit)
Layout to Wish	Circuit Simulation	Timing Analysis

TABLE 1. Levels of design and verification.

At the architectural level, functionality is verified through functional simulation, while the circuit performance is verified through timing analysis. The functional and circuit models that are used for design verification are automatically synthesized from the design specifications. Typically, the major portion of the development time is spent on these two verification activities. Because the system synthesizes the simulation and timing models automatically as part of module compilation, the design engineer may move with relative ease between design specification and design verification. This ease encourages a design method that is a process of successive, or incremental, refinement.

Separation of Functional Simulation and Timing Analysis

Table 1 shows the conventional levels of IC design and their respective verification operations. As mentioned above, most of the verification efforts in conventional IC design are concentrated on checking the equivalence between the various design levels. Silicon compilation, on the other hand, replaces all of the manual steps between architecture design and layout design, thus obviating the associated equivalence checking. Instead, this design process requires only functional simulation at the architectural level (to verify the architectural design) and timing analysis at the layout level (to verify the timing performance).

RCL000100

Designers commonly handle functional and timing designs separately, even though the two are to some extent intermingled in an iterative process. When working with synchronous circuits, it is often possible to treat function and timing as separable processes, and thereby deal with only one variable at a time. We strictly separate functional simulation and timing analysis functions in the Genesil system, because the two differ markedly in their goals and methods.

Because a large number of test vectors typically are required to adequately verify all the functions of a large digital network, simulation can be rather time-consuming. Therefore, it is extremely important to minimize the compute time required for each cycle of simulation. For this reason, simulation should be done at the highest possible level (without sacrificing accuracy). Computing timing delays concurrently with the evaluation of functional behavior not only may not yield absolutely accurate timing results due to circuit interactions, but also may further burden the computational requirements for each simulation cycle. While the test vectors are driving the functional simulation to exercise all the functions of the circuitry, timing delay paths could be computed redundantly if they were tagged along in the calculations. On the other hand, a dedicated timing analysis program that evaluates each timing delay path only once would be much more efficient in terms of compute time.

While the compute-time resource is a problem when many test vectors must be used to verify function, the engineering resource in generating those vectors is an even bigger problem. When a design engineer is running a functional or logic simulator, he focuses on searching out the problems in the function of the design. Since his main concern is to design the test vectors for that purpose only, the test vectors may not exercise all of the critical delay paths in the circuit. Hence, some critical short paths (race conditions) and long paths (delay times) may go unnoticed.

On the other hand, a timing analysis program can exhaustively analyze all possible timing paths in the circuit. With pruning of irrelevant paths by the designer, this approach has been found to be feasible. In contrast, it is not feasible to generate automatically test vectors for functional simulation, because it is not possible for a machine to know the intent of the design.

Another distinction argues for the separation of functional and timing analyses: Whereas functional simulation can usually be performed without reference to the process technology, timing analysis calculations depend in detail on physical parameters.

Functional Simulation

For a large digital network, a number of test vectors on the order of 100,000 is not uncommon. If the circuit is modeled at the gate or switch level, the CPU run time and memory required for each cycle of simulation are very high. Table 2 quantifies this statement for one instruction cycle in a VLSI chip (Kleckner *et al.* 1982).

Silicon compilation focuses the process of designing VLSI chips towards the architectural level, which can be conveniently simulated with functional models. Typically, the majority of a chip design would be composed of complex functional blocks, while the "odds and ends" would be implemented using gate-level blocks. The blocks that are defined at the functional level do not have to be simulated at the gate or switch level, because functionality at these levels is guaranteed by the compilation process. Performance verification, one of the classical imperatives for low-level simulation, is accomplished in an entirely separate step, via timing analysis, as discussed below.

Level of Simulation	CPU Time	Memory (MB)
RTL	<<1 min	<<0.5
Logic	10 min	4
Timing	8 hours	30
Circuit	6 months	250

TABLE 2. Comparison of run time and memory requirements for simulation of one instruction cycle in a VLSI chip.

Genesil's functional simulator is implemented using the selective trace event-driven technique, which permits multiple system clocks. Signal values are evaluated between clock edges; in other words, the clock cycles are assumed to be long enough so that all signals stabilize before the clock edges. (The minimum clock cycle times are determined in timing analysis.) Signal values are evaluated by calling block models that dynamically interpret their respective functional parameter specifications.

The simulator's user interface enables the designer to observe and/or modify the state of the simulation network and environment by defining or deleting clock cycle references; setting up and invoking checkpoints; defining vectors and mnemonics; issuing general resets; loading or clearing the contents of a RAM, ROM, or PLA array; and setting up test vectors. In addition, an interactive screen interface can trace signal histories and allow the user to observe and alter current network states. Various screen formats may be defined by the designer for use in interactive simulations or in viewing results of batch-mode runs.

Timing Analysis

Timing performance in conventional VLSI design commonly has been determined by extracting all the parametric details of the layout of a circuit and then feeding those parameters to a circuit simulation model of the portion of the circuit under study. Table 2 indicates the exorbitant amount of compute time needed to simulate VLSI chips at the circuit level. The magnitude of this expense makes it impractical to use circuit-level simulation to obtain timing data, except for small and isolated pieces of circuitry.

Therefore, the design engineer usually identifies manually portions of the circuit whose delay times he deems to be the worst cases. Due to the complexity of VLSI designs, such manual efforts tend to miss some critical paths, either because of oversight in path selection or mistakes in the circuit design. Furthermore, circuit simulators are data-dependent; the timing information produced depends on the input stimuli. To eliminate these problems, analysis must be performed on the entire chip in a data-independent fashion. However, in order for such automatic path enumeration to be practical, the timing analysis must run much faster than circuit simulation does.

Several timing analysis algorithms have been reported recently (Jouppi 1983; Ousterhout 1983; Lin *et al.* 1984). In terms of

RCL000101

CPU time, a timing analysis program runs up to 10,000 times faster compared to a circuit simulation program such as SPICE, while yielding results that are very close to those that a circuit simulator would estimate for most delay paths.

Because of their speed, timing analyzers can be used for checking all possible timing paths in a VLSI chip. Unlike a circuit simulator, which is driven by external stimuli provided by the design engineer, a timing analyzer automatically enumerates all possible timing paths, analyzes their timing delays, and reports on the worst of them. Hence, this approach is not vulnerable to the design engineer's fallibility in picking out critical paths.

Although timing analysis is not as accurate as circuit simulation, through careful crafting of the device models, adequate results can be obtained. For example, the Genesil Timing Analyzer is fast (2000 transistors per minute) and accurate ($\pm$ 10% of SPICE results). The program requires no test vectors, and is exhaustive in its path analysis.

However, in general, not all of the topologically possible delay paths represent signal paths that can occur in practice. Therefore, a timing analysis system must provide a user complete control to filter out unwanted or illogical results. In setting up a timing analysis in Genesil, the user may bind nodes to fixed values, force the direction of signal flow on bidirectional wires, flag nodes to be ignored, and hard-wire delays between two nodes.

To obtain accurate timing results, routing should be completed on the module or chip, because interconnection impedances significantly affect timing. However, the timing of portions of a module may be analyzed as it is being composed by estimating the external load impedances. To do so, a routing estimation is performed or else estimated models of the external circuitry are provided. Such estimations can later be confirmed by analyzing the finished module.

When timing analysis of a synthesized block is requested, the analyzer examines every possible path through the circuitry (subject to the user constraints mentioned above) to determine the worst-case delay times. The system uses built-in knowledge about which parameters are most important to produce a timing data-sheet, similar to what one would see in a typical component "data book". Several reports are provided:

1. A *timing* report identifies those paths that limit the minimum clock periods and duty cycles (rank-ordered from worst to best), lists the corresponding clock period and duty cycle values and the total cycle time, displays the input set-up and hold times, and lists the output delay times with respect to a clock transition.
2. A *node-delay* report lists the maximum and minimum delay times for a given node with respect to the clocks.
3. A *delay-path* report lists the paths that set the maximum and minimum delay between any two nodes.

If, after reviewing the timing report, the engineer finds unsatisfactory delays in portions of the circuit, he can attempt to refine the design. He may rearrange the block placements in order to shorten wiring lengths of critical signals. Or, he may redesign the circuit. For example, long delay paths can be broken by latches, creating a pipeline.

The Genesil Timing Analyzer works from a timing model that is synthesized by the system at the time the block or module is compiled. The model consists of a representation of the circuit at the transistor level, including all intrinsic resistances and capacitances, and those parasitic values contributed by the interconnection network.

As a result, the attributes of transistors and nodes consist not only of information about themselves, but also of information about their environment and how they are used. For example, the model resolves the directions of all signal flows, and classifies all transistors into one of several types, such as precharging transistors, bootstrap drivers, discharging transistors, or pull-ups. Clearly, the customized device models are crucial to the accuracy of the timing analyzer and also depend in detail on the underlying technology. In order to ease the job of calibrating these models, they are stored in look-up tables (called a "fabline file") instead of being coded into the program algorithm. Thus, the timing analyzer is accurate in terms of the fabrication line specified for the block, module, or chip. By extension, if the system is calibrated for several lines, a designer can compare the predicted performance of a circuit from several prospective foundries by changing the fab-line parameter and recompiling.

Conclusions

Logic design is a process of incremental refinement, in which design specifications and design goals co-evolve. The design method embodied in the Genesil system requires the user to perform a functional simulation at the architectural level and subsequently, a static timing analysis based upon a model of the final layout. The functional simulation is efficient in terms of computer resources and "easier" than a logic simulation, because the number of structures being simulated in a typical VLSI chip will be less than 100, rather than in the tens of thousands. The static timing analysis runs much faster than a circuit simulation does and automatically enumerates all possible delay paths. Performance verification reduces to a process of checking whether a worst-case path satisfies the design specification. □

References

Jouppi, N. 1983. "TV: An nMOS Timing Analyzer." *Third Caltech Conference on Very Large-Scale Integration*. Computer Science Press, Rockville, MD.

Kleckner, J.E., R.A. Saleh, and A.R. Newton. 1982. "Electrical Consistency in Schematic Simulation." *ICCC*.

Ousterhout, J. 1983. "Crystal: A Timing Analyzer for nMOS VLSI." *Third Caltech Conference on Very Large-Scale Integration*. Computer Science Press, Rockville, MD.

Lin, T. and C.A. Mead. October 1984. "Signal Delay in General RC Networks." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

About the Author

Edmund K. Cheng is the Vice President of Engineering at Silicon Compilers, Inc. He received the BSEE degree from Ohio University and the MSEE and Ph.D. from the California Institute of Technology. For six years he worked in the microcomputer design engineering department at Intel, where he developed the A/D converters for single-chip microcomputers and managed automotive-engine-control projects. Since 1981, Ed has led the development of the Genesil silicon compiler.

RCL000102

C. .D Systems for IC De.ign

MARVIN E. DANIEL AND CHARLES W. GWYN, SENIOR MEMBER, IEEE

Abstract—As integrated circuit (IC) complexities increase, many existing computer-aided design (CAD) methods must be replaced with an integrated design system to support very large scale integrated (VLSI) circuit and system design. The framework for a hierarchical CAD system is described. The system supports both functional and physical design from initial specification and system synthesis to simulation, mask layout, verification, and documentation. The system is being implemented in phases on a DECSystem 20 computer network and will support evolutionary changes as new technologies are developed and design strategies defined.

Introduction

COMPUTER-AIDED DESIGN (CAD) of integrated circuits (IC's) has had various interpretations as a function of time and definition source. These interpretations range from use of simple, interactive graphics and digitizing systems to individual programs used for circuit or logic simulation, mask layout, and data manipulation or reformatting. In many instances, the word "aided" is deemphasized to imply nearly automatic design. Within the context used in this paper, CAD refers to a collection of software tools to provide the designer with design assistance during each phase of the design. Although many decisions are made by the software during the design process, important decisions are the designer's responsibility. The computer aids or tools provide the designer with a rapid and orderly method for consolidating and evaluating design ideas and relieve the designer of numerous routine and mechanistic design steps.

Background

A brief review of some of the techniques and terminology that evolved into today's CAD is instructive to viewing the collection of tools available today. CAD had its origins in the mid-to-late 1950's. With the advent of high-speed digital computers, some of the pioneering work of Kron [1], [2] was applied to the simultaneous solution of network equations. This formulation was used, in part, by computer codes such as NET-1 [3] (Network Analysis Program) and ECAP [4] (Electronic Circuit Analysis Program). From a physical point of view, solution of the network problem predicts the behavior of a system in terms of the element characteristics and interconnections. Viewed as a mathematical problem, the properties of a topological structure (linear graph) and a superimposed algebraic structure (interrelations of the nodes, branches, and meshes of the graph) are determined.

Manuscript received April 10, 1981; revised September 8, 1981. This work was supported by the U.S. Department of Energy.

The authors are with Sandia National Laboratories, Integrated Circuit Design, Dept. 2110, Albuquerque, NM 87185.

This early work of solving steady-state and transient netwo problems was the genesis of circuit simulation and was des nated CAD. By the early 1970's, the growing need to simula large circuits and thus obtain the detailed solution of a lar number of partial differential equations forced the develo ment of optimization methods and higher levels of abstracti to represent physical systems. Whereas very detailed mod are used to simulate the behavior of individual transisto more abstract representations are used for timing and lo simulation. A timing simulator uses only current-volt tables for transistor models; capacitive loading, and circ connectivity to determine the signal waveforms at each c cuit node. Logic or gate-level simulation solves the equi lent Boolean equations with delay elements inserted betwe gates to account for signal timing.

The use of computer aids in the layout of IC masks ess tially proceeded along two approaches: interactive graphic sy tems and automatic layout based on standard cells. Early i teractive graphic systems provided a method for capturing d design by recording coordinate information by manually di tizing and editing data. Methods for superimposing mi layers, scaling, enlarging, contrasting, and reviewing the resu were expanded to provide fast, sophisticated drawing com mands for constructing, editing, and reproducing complex l ures; performing dimensional tolerance checks; selective era expand, move, and merge; symbolic input; pattern generati etc.

Automatic layout methods have classically relied on a libra of circuit components or cells in the form of mask geometri defining logic gates. Most software required cells with sta dard heights and varying width. The layout software, plac cells in rows, attempts to optimize the cell position in the ro and interconnects the cells in wiring channels between the ce rows. The automatic standard cell layout programs ha evolved from simple linear cell placement in a single ro which was subsequently folded to fit a square area [5], complex placement in two dimensions. The early standard ce layout aids supported the use of single entry (connections o one side of the cell) cells placed in the row in a back-to-ba configuration. Power was distributed to the cells through d common connection on the backside of the cell. Newer sta dard cell layout programs support cells containing terminal on both sides of the cell. Instead of placing cells in a back-t back configuration, each cell row contains a linear placeme of the cells [6], [7] and is separated by wiring channels.

The early use of CAD for physical design verification con sisted of performing simple design rule checks for width an spacing violations on mask artwork files. Functional integri was verified through circuit simulations.

0028-0070/82/0100-0002$00.75 © 1982 IEEE

RCL000103

Initial Sandia Design Aids

After the initial decision to establish a CAD capability for integrated circuits at Sandia, software was obtained from universities and private industry wherever possible to provide a basic capability. For example, the SPICE [8] circuit analysis code was obtained from the University of California at Berkeley, and the PR2D [6] standard cell layout program for metal gate cells was obtained from RCA. Since Sandia had previously developed a simple logic analysis capability, this work was accelerated to provide gate-level simulation. Software was developed to postprocess the mask layout data to pattern generator formats for mask generation and plot file information for the Xynetics flatbed plotter. A software package was purchased from Systems, Science, and Software, Inc., and installed on an existing interactive graphic system. In addition, translator subroutines were written to convert design information data formats to minimize the manual data translation and to allow the design information to be added only once in the system. The above process is oversimplified, since a substantial commitment of staff was required to: (1) understand the acquired software and debug and correct errors, (2) perform modifications to support the software on Control Data computers, (3) identify and develop required translator software, and (4) develop additional design aids. These problems and the subsequent software modifications required a large manpower investment, since most of the software was not documented, had evolved over many years with several authors, used nonstandard Fortran, and was unstructured.

CAD System Requirements

Although the initial design aids provided a valuable capability for simple metal gate CMOS custom IC design, many deficiencies were identified. These deficiencies coupled with the need to support new technologies with shrinking design rules (and thus rapidly increasing circuit complexities), new design styles, and changing design objectives, required a new approach to the development of aids for IC design.

Several general objectives for new aids were identified. The new aids must (1) be user oriented, (2) use modular software, (3) be evolutionary to meet changing design needs, and (4) be integrated into a complete design system rather than exist as an independent collection of disjoint tools.

Computer aids must support both functional and physical design. Functional design aids include synthesis, verification, simulation, and testing at architecture, system, logic, circuit, device, and process levels. Physical design aids support partitioning, layout, and topological analysis at all design levels. Functionality, testability, and physical design must be considered in parallel throughout the design process.

All CAD software must be as technology independent as possible and support various levels of designer sophistication from inexperienced first-time users to state-of-the-art system designers. To meet these goals, the interaction between individual programs, data base, and design engineer must be through a single, consistent interface. This interface should support monitoring the progress of a design, supply options at any stage in the design process, and assist in design documentation and maintenance.

TABLE I

Complexity	Examples
VLSI ($\sim 10^5$ devices)	Microcomputers, cryptographic circuits, ROMs, RAMs
LSI ($\sim 10^4$ devices)	Microprocessors, A/D & D/A converters, ALUs, FFT circuits, ROMs, RAMs
MSI ($\sim 10^3$ devices)	Adders, complex gates, multiplexers, ROMs, RAMs
SSI (~ 100 devices)	AND, OR, NAND, NOR gates, buffers, memory cells
Primitive elements	Transistors, resistors, capacitors

A system design language—or Hardware Description Language (HDL)—must be available for describing all levels of system behavior and structure. Organized in a hierarchical manner, the language should support functional and physical descriptions and the relationships among entities.

Synthesis aids must facilitate the addition of sufficient detail to generate a complete system description. Design verification software monitors internal consistency and completeness of system specifications. Final system specifications should be retained in a dynamic data base providing files in the proper format for input to each of the design aids and for documentation.

A complete CAD system satisfying the above requirements has been designed and implementation is in progress. The basic system description has been divided into three sections. The first section describes design flow and the concept of top-down design with bottom-up implementation. A hierarchical design approach is used with appropriate merging of levels to accomplish the design of VLSI systems. Requirements for specific computer aids are outlined in the second section. The final section describes the implementation philosophy, computer hardware, and present software development status.

Hierarchical Design

The CAD system supports a number of functions at each level in the design hierarchy. Design proceeds in a top-down sequence with bottom-up detailed implementation and addresses both functional and physical problems at each level.

Architecture, the top level of the design hierarchy, contains elements for a broad functional system description, requirements for interfacing units specifying performance and compatibility, and methods for partitioning the system into major functional blocks such as processors, memory, and I/O. The architectural specification can be expanded at the system level to generate a more detailed description consisting of register transfer level subsystem functions.

At the logic level, system functions are defined as interconnections of fundamental gates or modules. Logic modules can be further decomposed into circuit primitives (e.g., transistors, resistors, capacitors). Finally, in order to construct circuit elements and determine their behavior, the physical implementation and process technology may be considered.

To support a variety of technologies in a hierarchical design structure, a data base consisting of a library of elements of varying sophistication must be maintained. A distinct library must exist for each technology used. Typically, the hierarchi-

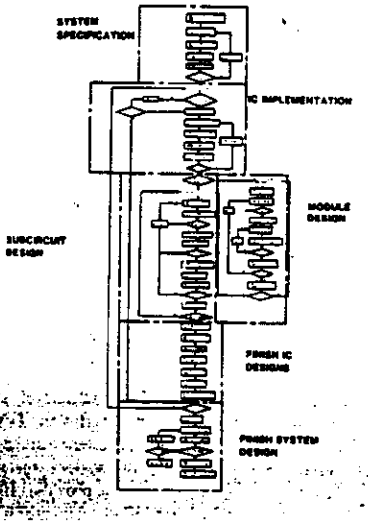


Fig. 1. System design sequence.

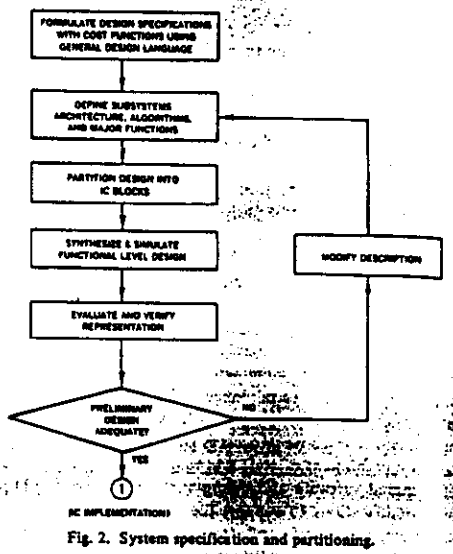


Fig. 2. System specification and partitioning.

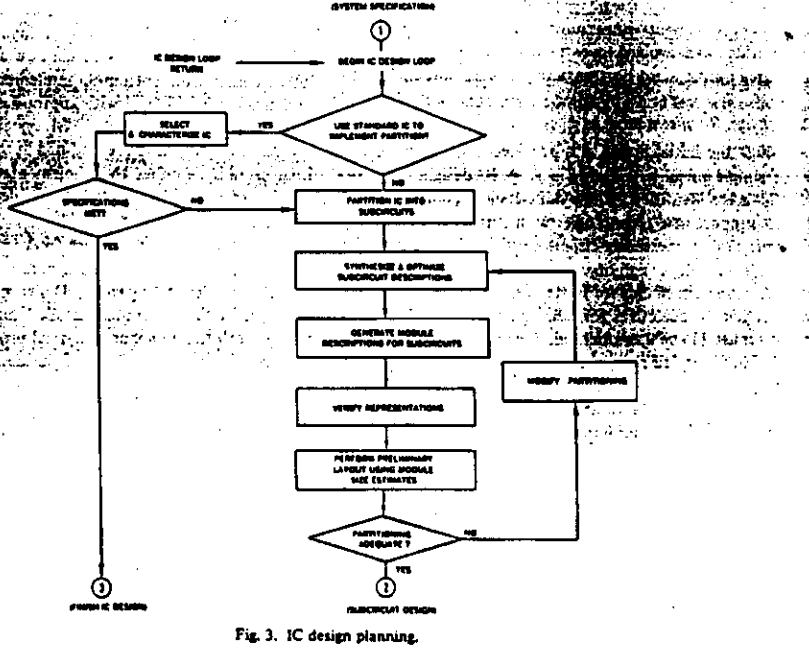


Fig. 3. IC design planning.

cal library will contain circuits and subcircuits at various levels of complexity (VLSI, LSI, MSI, SSI, circuit primitives and/or discrete devices) as defined in Table I.

Electronic system design using computer aids can be divided into six major design sections: (1) system specification and partitioning, (2) IC design planning and initial implementation, (3) subcircuit design, (4) module design, (5) completion of individual IC design, and (6) completion of system design. A design flow diagram summarizing these major functions is shown in Fig. 1.

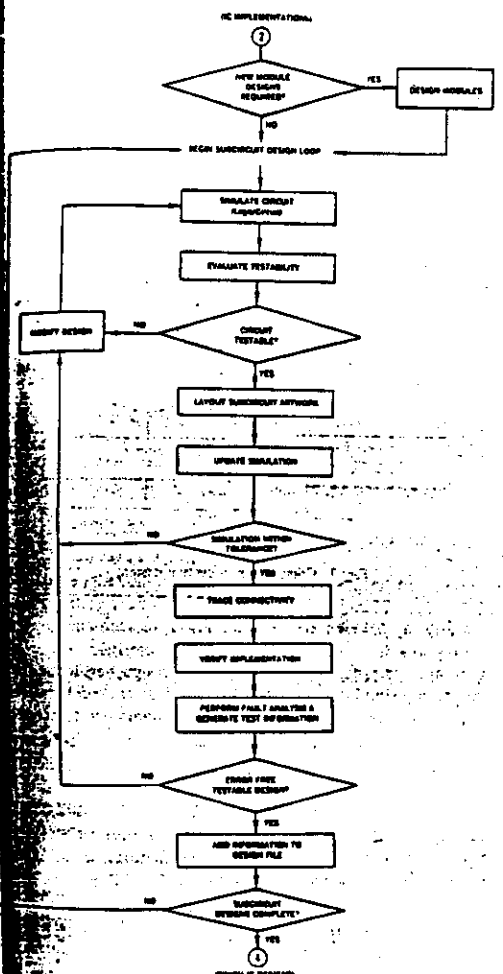


Fig. 4. Subcircuit design.

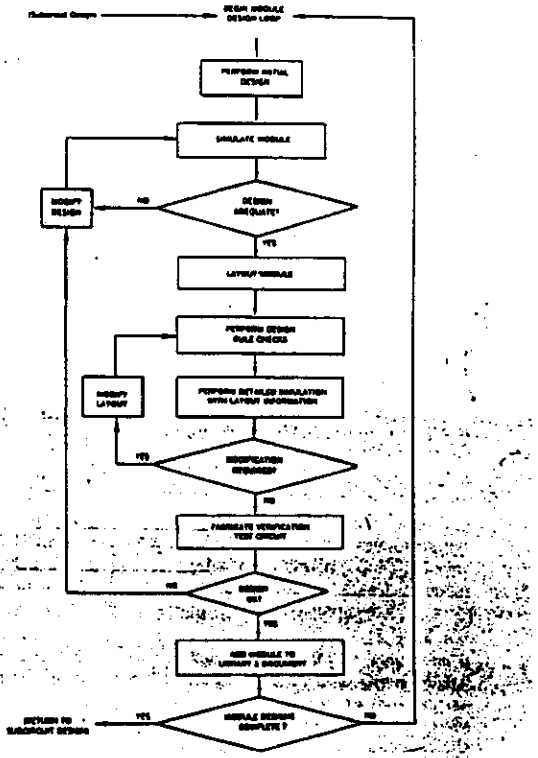


Fig. 5. Module design.

The flow diagram in Fig. 1 is subdivided and enlarged to show the individual design steps in Figs. 2–7. The first major step consists of developing system specifications and partitioning guidelines (Fig. 2). After developing general specifications, subsystems, architecture, algorithms, and major functions are defined. Each subsystem is partitioned into IC blocks which are synthesized and simulated at the system design level. This preliminary design is evaluated and modified if necessary by repeating the above steps.

The next major step in the design is implementation of each block based on the system specification (Fig. 3). For a general electronic system, the first decision to be made is whether custom or commercial circuits will be used. If a characterized commercial circuit is available in the library, the circuit is selected. If the required commercial circuit is not in the library, the specifications must be obtained and compared with the system specification. If the system specifications are met, the commercial circuit can be used; if not, a custom circuit must be designed.

The first step in designing a custom IC is partitioning the IC into subcircuits. Each subcircuit is synthesized and optimized. Module descriptions for each subcircuit are developed and verified, and a preliminary layout using the estimated module sizes is performed. At this point, a decision can be made concerning the adequacy of the partitioning. If the preliminary layout does not conform to the size restrictions for the circuit or if during the circuit verification, signal-delay paths are longer than desired, the partitioning must be modified and the circuit design repeated.

If partitioning is acceptable, the subcircuit design step can be initiated (Fig. 4). For each subcircuit design, a decision must be made concerning the adequacy of modules in the library for implementing the design; new modules must be designed and characterized as required (Fig. 5).

During subcircuit implementation, each circuit can be simulated at various individual or combined simulation levels, the testability evaluated, and the artwork generated. After art-

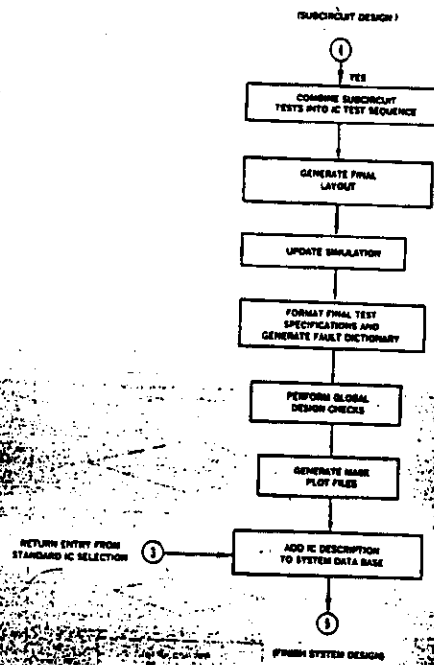


Fig. 6. Completion of IC design.

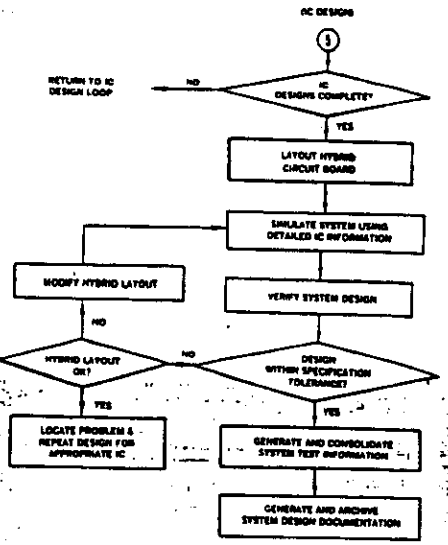


Fig. 7. Completion of system design.

work design, the simulation net list must be updated to include circuit parasitics associated with the specific layout. This simulation also provides information concerning circuit output drive and power required for subcircuit operation. If the simulation is within tolerance, static design verification is initiated; connectivity is traced; and the layout is checked for conformance to design rules. The layout and associated simulation characteristics are compared with the initial circuit specifications. Finally, test information is generated for the subcircuit.

After each subcircuit design has been completed and verified, the circuit design information is added to a design file in the data base. The next major step consists of completing the IC design by combining subcircuit design information (Fig. 6). The IC layout is assembled by placing and interconnecting subcircuit layouts. The simulation is updated to assure correct circuit operation, test specifications are generated, and a fault dictionary is developed for system diagnosis. Physical design verification, including global rule checks and connectivity checks, is performed, and the mask information for each IC is generated. Finally, design information for each IC is added to the system data base.

The last major step in the design sequence consists of completing the system design (Fig. 7). After selecting commercial circuits or designing custom IC's, layouts are performed for the system circuit boards. After the printed- or hybrid-circuit board layouts have been completed, the entire system is simulated, using detailed design information for each circuit. This simulation determines the electrical characteristics of the system and provides a final functional design verification by comparing simulation results with initial design specifications. Test sequences for the individual IC's are combined to produce system tests at the printed-circuit board or hybrid-circuit level. During the final step, system design documentation is generated and archived. Generating system documentation consists of obtaining information from the design data base and consolidating it into the appropriate format. In addition to the IC mask plot files and test-sequence information, the documentation includes system performance information with tolerance specifications for acceptance of the final system.

The complete CAD system for IC design will include a number of computer programs. These aids can be categorized as follows: design specification and partitioning, system and circuit synthesis, system partitioning, simulation at various levels, IC mask layout, design verification, testability evaluation, test sequence generation, data base, and design documentation.

Specific Computer Aids Requirements

In addition to a number of specific computer programs for supporting a hierarchical design, an extensive data base and support software for monitoring the design flow and communication are required. Requirements for specific programs and interfaces are outlined below.

Design Executive

Currently, most design automation programs exist as independent entities with idiosyncratic user interfaces and incompatible I/O structures. A Design Executive can provide a sing

consistent interface between a user and the complete set of computer aids available on the CAD system.

The Design Executive supports the complex CAD system structure by mediating interactions between the user and the system, the system and the data base, and the user and the data base. The executive system user interface supports comprehensive "HELP" capabilities, the HDL, and a logically integrated command language. Documentation describing the system and the CAD codes is maintained by the Design Executive in a hierarchical configuration. User access to this documentation structure is facilitated by the intelligent intervention of the Design Executive.

HDL–Design Specification

The structure and behavior of a digital system must be described in various ways at many levels to completely characterize a design. Existing "languages" such as ISP [9], DDL [10], SDL [11] and are usually associated with specific descriptions of architecture, system behavior, system structure, logical structure, circuit structure, logical behavior, or physical structure. These descriptions lack the generality required to support VLSI design, since most are applicable to one level of design and a particular design style and are not integrated into a complete design system.

Ultimately, a comprehensive HDL must be used to describe all levels of system behavior and structure, including normal and faulty electrical operation, logical and functional behavior, and physical structure. Organized in a hierarchical manner, the HDL should allow description of functional and physical entities and relationships among entities.

Synthesis and Functional Verification

Functional synthesis begins at the architecture level with a description of the overall behavior of a large system and interaction with the environment. As synthesis proceeds, more structural detail is added at system, logic, and circuit levels in the form of interconnection structures. Concurrently, behavioral specifications are developed at each level of the design hierarchy.

During synthesis, the actual behavior of the system must be forced to match the specifications. To effect the match, models are required for computational algorithms and interpreters, and procedures for mapping one into the other. Models must be expressible in a computer-readable form in order to manage complexity with an interactive computer support system, permit the separation of structure from associated behavior, and support direct fabrication of the synthesized system.

Simulation

Simulation, or dynamic design verification, is the process of calculating the behavior of a system within an environment specified by the designer. The objective of simulation is to verify that the system will perform correctly in an operational environment.

Ideally, simulation should be multilevel; i.e., consider larger circuits at a less detailed level with the capability of simultaneously simulating subcircuits more exactly. This concept complements the hierarchical design implementation. In a hierarchical simulator, the basic elements are more complex than simple modules or gates. Models are formulated for entire IC's and include gate, function, and transistor models. Parts of a system are simulated in great detail while other parts are simulated abstractly. Models are written in high-level languages, with models for MSI or LSI blocks only slightly more complex than the model for a simple NAND gate. Thus a system of many thousands of devices or gates can be reduced to a few hundred hierarchical models, making it feasible to simulate entire VLSI systems.

Testability

Testing is the process of verifying that a system is operating properly. Because of the increased complexity of VLSI circuits, the difficulty of testing is substantially increased. To ensure that a circuit can be tested, design-for-testability techniques must be used throughout the design process.

Approaches to design-for-testability fall into two major categories: testable design styles and testability measure analysis. The use of a testable design style guarantees that generating tests for a circuit will not be impossible. Testability measure analysis computes the difficulty of controlling and observing each internal node from primary input or output pads. This information is used in the design process to locate potential circuit testing problems and provide feedback about the effect of circuit modifications on testability.

In developing a test sequence, knowing that a good test exists is not the same as knowing the test. Deriving a test is the task of the test sequence generation phase. Given a digital circuit and a set of possible faults, a series of input signals (vectors) is generated that will force any faulty circuit to behave differently from the fault-free circuit. The test sequences depend not only upon the intended function of the circuit but also upon the fault set assumed. Test-sequence generation must proceed concurrently with the hierarchical design process.

Physical Design Aids

Physical design aids include tools to partition, place, and interconnect circuit components and verification tools to ensure the synthesis has been performed properly. To provide an optimum system, physical design must be considered in parallel with functional design and testability.

Partitioning–Partitioning programs operate on both functional and physical entities. Partitioning techniques must function in both the initial design and detailed implementation phases. During top-down design, partitioning aids are needed for hierarchical decomposition. During bottom-up implementation, partitions are modified based on lower level implementations.

In addition to optimizing a circuit element assignment based on size and external pin connections, parameters such as circuit speed, power dissipation, and functional groupings must be considered. The partitioning aids must be capable of optimizing any of these quantities as a function of the others.

Symbolic layout–Provides a shorthand method for manually sketching a circuit layout using specified symbols to represent

various types of transistors and interconnections. The initial layout can be performed on grided paper or directly on an interactive graphics system. During layout, the designer is not concerned with the geometric design rules. Symbolic layouts are postprocessed using computer aids which expand and relocate all symbols and interconnects based on the geometric design rules for the specified technology to provide mask artwork files.

Physical layout—The physical layout phase of IC design involves positioning and interconnecting electrical components. In hierarchical design, the concept of a component is generalized. Components may range in complexity from primitive transistors and fundamental gates to microprocessors that can be combined to form a microcomputer.

Hierarchical layout is applicable to a wide range of physical design levels. At the highest design level, the shape of LSI components may be adjusted and combined to form a VLSI circuit. At the lower design levels, the same algorithms may be used to combine gates into registers or transistors into gates.

Topological analysis—Because of the high costs and long lead times involved in the fabrication of IC's, it is important to verify the correctness of a circuit design before manufacture. Verification tools must ensure correct physical mask layout, functional operation, and that electrical characteristics are within specified tolerances.

Topological analysis tools can be used to verify correctness of physical layout data by examining the artwork data and the circuit inputs and outputs. Design-rule checking codes perform geometrical, logical, and topological operations on artwork data and compare the results with design rules for the specified technology. Connectivity and electrical parameter data are used to reconstruct a detailed circuit including parasitic electrical components. This reconstructed circuit can be used in performing an accurate functional and timing analysis.

Circuit Design Documentation

After a circuit or system design has been completed, the design can be documented by collecting information from the data base generated during the design process. Design documentation includes information for fabricating the IC's and system as well as information for archiving the completed design to support later modifications. Computer aids should collect manufacturing and archival information and generate appropriate files, specifications, and reports.

Data Base

In general, the distinction between programs and data is that programs are active, data is passive (i.e., programs operate on data). The necessity for supporting the initial design, design changes, and providing software transportability dictates consistency in data handling procedures and mechanisms. Therefore, an efficient data base is vital for coordinating the various functions. As systems become more complex, it is imperative that an overall data base be established and maintained to ensure consistency in design and to eliminate duplication of information.

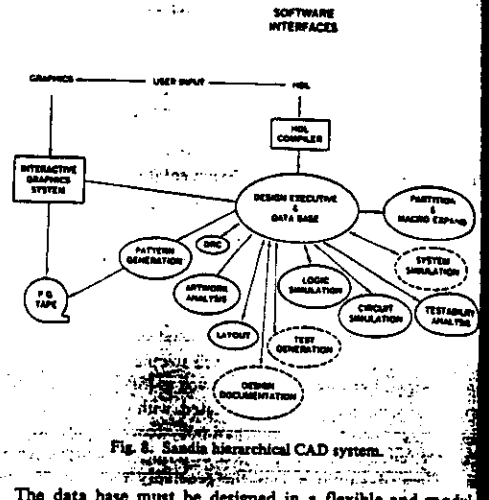


Fig. 8. Sandia hierarchical CAD system.

The data base must be designed in a flexible and modu fashion to provide upward compatibility with computer ha ware and the evolving CAD software. Modularity is importa where various portions of the data base may reside at m than one computer node. In addition to the actual data u by CAD programs, the data base should contain all necessa information to properly document elements being designed.

Sandia CAD System Implementation

A CAD System meeting the criteria outlined above is be implemented in phases at Sandia National Laboratories to p vide design capability for LSI circuits initially, with evol tionary expansion and enhancement as required to suppo VLSI designs.

The design procedures consist of: (1) identifying critical a in the design process based on user needs, (2) acquiring exi ing software when available and subsequently modifying it meet system and user needs, (3) developing new aids wi appropriate expansion capabilities to support hierarchical d sign, and (4) integration of all aids into the design syste framework. A block diagram outlining the present system i plementation is shown in Fig. 8. The solid blocks repres areas where design support is presently available, and t dashed blocks indicate work in progress. Brief descriptions the hardware and software are given below.

Hardware

The software is supported on a dedicated DECSyste computer system containing 1 million words of main memo 160 million words of disk files, and two 9-track 800/16 bit/in, 75 in/s tape drives. Two Applicon 860 interact graphics systems and a Versatec 36" electrostatic plotter connected to the DECSystem 20. A VAX 11/780 comput containing 4 megabytes of main memory, 323 megabytes disk files, and 2 9-track 800/1600 bit/in, 125 in/s tape dri is interfaced in a network configuration to provide additio computing capability. Communication among the comput

and the Applicon systems is supported by DECnet. Both dial-up and direct-wired access to the system provide support for alphanumeric and graphics terminals.

Design Software

A Universal HDL (UHDL) is being developed in essentially three stages consisting of: (1) language formation by phrase concatenation, (2) redundancy removal, and (3) consolidation and rewrite of the compound language. During the first step, the SDL [11] language (a PASCAL-like description) is used to describe system structure. Other languages are appended in phrase form to describe functional behavior. After gaining experience with the composite languages and after the deficiencies and required enhancements have been identified, a language for the initial UHDL will be written. Continuous refinement and enhancement of the UHDL will be required to meet design and documentation needs.

Translation from UHDL descriptions (initially SDL) to the required formats for simulation, layout, and verification programs is performed by the Design Executive. The Design Executive also manages the data flow between the Data Base files (outlined below) and each of the design aids.

Design synthesis aids currently available include MINI [12] and DDA [13] (Digital Design Aids). MINI, a Programmable Logic Array (PLA) minimization program uses a branch-and-bound algorithm to produce the optimal two-level realization of a set of Boolean equations. DDA provides synthesis of a sequential-state machine based on flow diagram, state assignment, and flip-flop type.

Logic-level design consists of defining, in complete detail, all components (gates, cells, modules, etc.) and interconnections required to implement a specified function. The SALOGS [14] (SAndia LOGic Simulator) program performs true-value 4-state logic simulation (true, false, undefined, and high impedance) and 8-state timing simulation (four states plus transition to each of the four states) as well as states-applied and gate activity analysis and fault simulation. SALOGS is used for dynamic design verification, test sequence evaluation, and fault coverage calculations. Race and hazard analysis is available and the program is capable of using library or user-defined models.

A typical input/output for SALOGS is shown in Fig. 9 for a small logic circuit. The input consists of a connectivity description of logic gates or cells contained in the model library. The plotted output describes the logic signal changes as a function of time for each of the nodes indicated.

Although approximate timing simulation is accomplished in SALOGS through the use of extra states to represent logic values in transition, more exact timing simulation is possible with MOTIS-C [15] and SIMPIL [16] for MOS and I^2L circuits, respectively. These programs use tables to describe active devices in a circuit instead of models with complex equations. Both programs provide more accurate analysis than logic simulators and run more efficiently than circuit simulation codes such as SPICE.

A more exact circuit analysis is obtained by using the SANCA (SANdia Circuit Analysis) program circuit simulation program based on SPICE. This program simulates MOS and bipolar

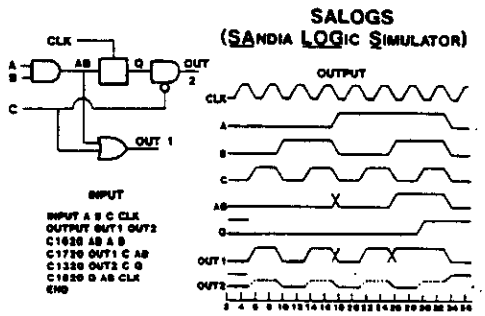


Fig. 9. Typical logic simulation input and output using SALOGS.

circuits, including analog and digital circuits, by numerically solving the network equations to calculate the desired voltages and currents. The cost of increased accuracy (over timing simulation) is a decrease in both execution speed and circuit size that can be analyzed. Model libraries [17] for circuit analysis are maintained to reflect varying levels of accuracy and different modeling philosophies. Circuit models based on device physics, empirical behavior, and hybrid approaches are all used in the circuit simulation environment. SANCA is an enhanced, quasi-interactive, topological analysis program containing model information for specific Sandia technologies and output plot routines.

Physical mask layout is performed using the SICLOPS [18], [19] (Sandia IC Layout Optimization System) and SLOOP [20] (Standard cell LayOut Optimization Program) programs to automatically place and interconnect generalized circuit elements. Two different forms of mask layout are used. A hierarchical mask layout system has been developed and used for specific designs. Hierarchical layout is applicable to a wide range of design levels. At one extreme, LSI components can be combined to form a VLSI circuit. At the other extreme, the same algorithms can be used to combine gates into registers based on a standard cell layout. This hierarchical approach relies on the use of SICLOPS to perform an automatic layout of an integrated circuit using rectangular-shaped blocks. The blocks must be rectangular and can be of arbitrary size and aspect ratio. The program automatically places the blocks and performs the interconnections. Each of the blocks can contain nested subblocks to any depth to provide a hierarchical layout. SICLOPS can also be used to layout a thick film hybrid circuit, as shown in Fig. 10.

The more conventional layout using standard cells placed in rows with interconnection between cell rows in routing channels is supported by SLOOP. In addition to designing modules used by SICLOPS, conventional standard cell IC layouts can be designed. The basic program can be used in an interactive mode to optimize the design, will perform a 100-percent completion of all interconnections by expanding the chip size as required to complete interconnections, contains hierarchical interfaces for use with the SICLOPS code, and supports multiport gates or cells. A typical IC design using SLOOP is shown in Fig. 11.

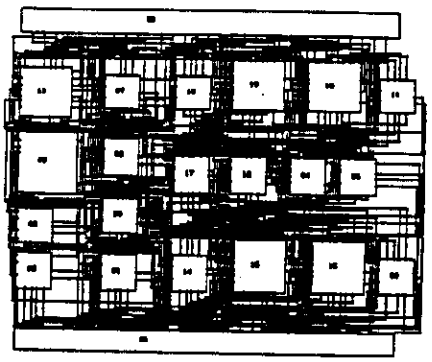


Fig. 10. Thick-film hybrid circuit layout obtained using SICLOPS.

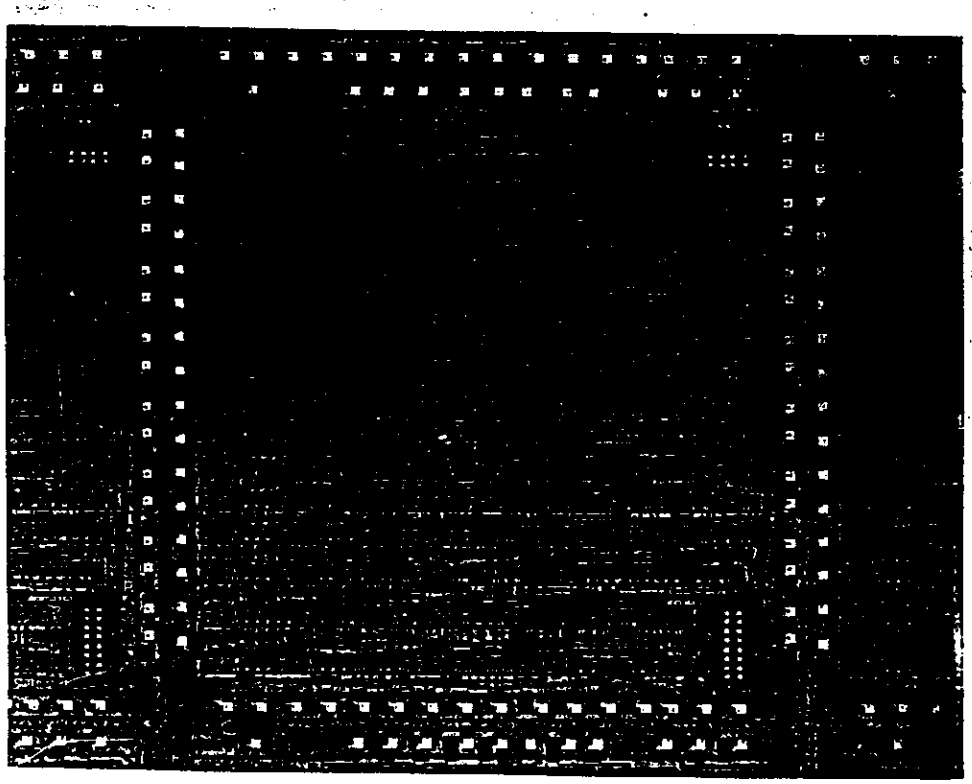


Fig. 11. Typical standard cell IC design using SLOOP.

The LOGMASC (LOGical MASk Checking) [21] program is a tool for design rule checking of IC masks. This program produces Boolean logic combinations of mask levels and is based on pattern recognition techniques. LOGMASC provides essentially all design rule checks which can be performed manually and provides pattern identification and extensive topological and geometrical information concerning the interrelationships of entities in the file. False errors are minimized by selectively applying design rule checks based on the circuit use of the particular patterns begin checked. Many of the algorithms can be used to solve other mask analysis problems in addition to the design rule checks.

During the circuit mask design process, transistor current or voltage gains, resistance, and capacitance values are scaled and converted to area definitions based on the nominal circuit parameter values used for circuit analysis. The actual sizes of the circuit elements and devices are adjusted to fit layout constraints based on minimum spacing, relative element partitioning, and size requirements. The actual circuit defined by the mask layout may be quite different than the original circuit schematic. For example, the interconnection distances between transistors introduce resistances and capacitances into the circuit which were not included in the initial circuit simulation. Additional parasitic transistors and diodes can be unintentionally introduced into the circuit which can cause improper operation. The CMAT [22] (Circuit MAsk Translator) code operates on circuit mask information, using pattern recognition to recognize and transcribe the mask areas into the circuit elements. CMAT employs the basic Boolean operations algorithms contained in LOGMASC and performs the required scaling operations to obtain circuit element values.

SCOAP [23] (Sandia Controllability Observability Analysis Program) is used to evaluate the testability of a circuit after the basic circuit design has been completed. This analysis is performed prior to layout and is based on a simple analysis of the logic interconnectivity. The SCOAP program calculates the ease or difficulty of setting an internal node from a primary input to a zero or one and the difficulty of observing a logic value at any internal node from a primary output. The zero and one values for controllability may be different depending upon the exact circuitry and interconnection of the logic gates. For nodes having very high values for controllability or observability changes are usually required in the logic circuitry to reduce the controllability and observability numbers. This reduction is required, since the numbers are proportional to the testing difficulty. For nodes with high controllability/observability measures, additional input or output test points or signal multiplexing may be required to set logic gates or observe node logic values to achieve circuit testability. At present, a structured design for testability, such as the IBM LSSD [24] approach, is not required. However, since the designer is responsible for designing a testable circuit, techniques equivalent to the LSSD approach may be used in the design based on the testability analysis information.

The testability analysis program is used for both combinational and sequential circuits. Combinational measures basically relate to the number of different line assignments or input assignments which must be made to set an internal node to a given value. For sequential circuits, the measures relate to the number of clock cycles or time frames required to propagate a specified logic value from an input pad to a node or from a specific node to an output terminal.

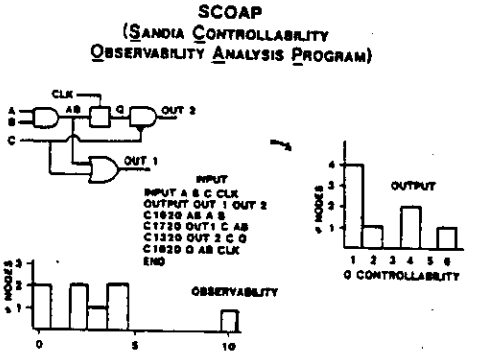


Fig. 12. Testability analysis using SCOAP.

An example of testability analysis is shown in Fig. 12. All input nodes are directly controllable as indicated for the four nodes with a "zero controllability" value of one. The most difficult node to control is usually an output node; for this example, output 2 has a zero controllability value of six. The observability is shown in the lower left of the figure. Again, the most observable nodes are the output nodes as indicated by the two nodes with an observability value of one. Ordinarily the most difficult nodes to observe are the input nodes and, therefore, the nodes *A*, *B*, *C*, and *CLK* should be the least observable. In this case, the *CLK* signal is input to a transmission gate between nodes *AB* and *Q*; a change of state at node *AB* must occur to observe the clock signal and, therefore, the *CLK* signal has the large value of 10 for observability.

Data Base

Because of the initial need for integrating loosely related CAD software and evolving design system specifications, a data-file base has been developed. The Sandia CAD data structure [25] is in the form of a "deciduous tree"; that is, one that can lose its leaves. This structure is a two-dimensional network with several restrictions. In one dimension the data structure forms a shallow tree one level deep. That is, any number of data types can be associated with a given node, but they may not be broken down further. In the other dimension, the structure is a network of nodes and subnodes to any depth. The leaves on the tree are the equivalent of small sequential files of data which can be retrieved by the user application program. This data file approach forces the designer to conform to a hierarchical structure while maintaining the freedom to store test files, documentation, etc., in the same file.

Summary and Future Plans

Although a basic computer-aided LSI design capability has been established and is used by design engineers, the system is continuously evolving as new capabilities and enhancements

are added. The modular programs and data structures, as well as the flexibility designed into the overall system framework, tend to minimize the cost for system modification as requirements change. In addition to continuous enhancement of existing programs, new aids for design synthesis, hierarchical simulation, symbolic layout, and test sequence generation will be developed. A continuing emphasis is placed on integrating the CAD software into a complete design system.

REFERENCES

[1] G. Kron, *Tensor Analysis of Network*. New York: Wiley, 1939.
[2] —, "A set of principles to interconnect the solutions of physical systems," *J. Appl. Phys.*, vol. 24, pp. 965–980.
[3] A. F. Malmberg, F. L. Cornwell, and F. N. Hofer, "NET-1 network analysis program," Los Alamos Rep. LA-3119, 1964.
[4] R. W. Jensen and M. D. Lieberman, *The IBM Circuit Analysis Program*. Englewood Cliffs, NJ: Prentice-Hall, 1968.
[5] A. Feller and M. D. Agostino, "Computer-aided mask artwork generation for IC arrays," in *Dig. 1968 IEEE Computer Group Conf.*, pp. 23–26.
[6] A. Feller, "Automatic layout of low-cost quick-turnaround random-logic custom LSI devices," in *Proc. 13th Design Automation Conf.*, pp. 79–85, June 1976.
[7] G. Persky, D. N. Deutsch, and D. G. Schweikert, "LTX—A system for the directed automatic design of LSI circuits," in *Proc. 13th Design Automation Conf.*, pp. 399–407, June 1976.
[8] E. Cohen, Program reference for SPICE 2, Memo ERL-M592, Electronics Research Laboratory, Univ. California at Berkeley, June 1976.
[9] M. R. Barbacci *et al.*, The ISPS computer description language, Tech. Rep Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA, Mar. 1978.
[10] W. E. Cory *et al.*, An introduction to the DDL-P language, Tech. Rep. 163, Computer Systems Lab., Stanford Univ., Stanford, CA, Mar. 1979.
[11] W. M. van Cleemput, A structural design language for computer-aided design of digital systems, Tech. Rep. 136, Digital Systems Lab., Stanford Univ., Stanford, CA, Apr. 1977.
[12] J. Abraham, MINI—A logic minimization program, Univ. Illinois, Urbana, IL.
[13] DDA—Digital design aids for state machine synthesis, Tektronix, private communication.
[14] J. M. Acken and J. D. Stauffer, "Logic circuit simulation," *IEEE Circuits Syst. Mag.*, vol 1, no. 2, pp. 3–12, June 1979.
[15] S. P. Fan *et al.*, "MOTIS-C: A new circuit simulator for MOS LSI circuits," in *Proc. IEEE 1977 Int. Symp. on Circuits and Systems*, pp. 700–703, Apr. 1977.
[16] G. R. Boyle, *Simulation of Integrated Injection Logic*, Memo. UCB/ERL M78/13, Electron. Res. Lab., Univ. California at Berkeley, Mar. 1978.
[17] D. R. Alexander, R. J. Antinone, and G. W. Brown, *SPICE 2 Modeling Handbook*, BDM/A-77-071-TR, May 1977.
[18] B. T. Preas and W. M. van Cleemput, "Placement algorithms for arbitrarily shaped blocks," in *Proc. 16th Design Automation Conf.*, June 1979.
[19] B. T. Preas and W. M. vanCleemput, "Routing algorithms for hierarchical IC layout," in *Proc. Int. Symp. Circuits and Systems*, July 1979.
[20] SLOOP (Standard-Cell Layout Optimization Program), Sandia Lab., to be published.
[21] B. W. Lindsay and B. T. Preas, "Design rule checking and analysis of IC mask designs," in *Proc. 13th Design Automation Conf.*, June 1976.
[22] B. T. Preas *et al.*, "Automatic circuit analysis based on mask information," in *Proc. 13th Design Automation Conf.*, June 19
[23] L. H. Goldstein, "Controllability observability analysis of digital circuits," *IEEE Trans. Circuits Syst.*, vol. CAS-26, Sept. 1979.
[24] E. B. Eichelberger and T. W. Williams, "A logic design structure for LSI testability," in *Proc. 14th Design Automation Conf.*, pp. 462–468, June 1977.
[25] J. D. Stauffer, "SADIST (The Sandia Data Index Structure), Sandia National Lab., Albuquerque, NM, SAND80-1999, Nov. 1980.

Marvin E. Daniel was born in Dexter, KS, on December 19, 1938. He received the B.S.E.E. degree from Kansas State University in 196[illegible], the M.S.E.E. degree from University of New Mexico, Albuquerque, in 1963, and the Ph.D. degree in electrical engineering from Oklahoma State University, in 1966.

In 1961, he joined Sandia National Laboratories, Albuquerque, where he has worked in a variety of positions including: design of test equipment, application of minicomputers to automated test systems, development of mathematical models for CAD programs, modeling of radiation effects in semiconductor devices, analysis of weapon systems, economic studies of fusion and fossil fuel energy systems, and for the past two-and-one-half years has been supervisor of the Computer-Aided Design Division in the IC Design and Fabrication organization. He is the author of numerous papers and articles, particularly in the areas of programs and models for computer-aided design.

Charles W. Gwyn (M'68–SM'75) received the B.S. degree in electrical engineering from the University of Kansas, Lawrence, in 1961, and the M.S. and Ph.D. degrees in electrical engineering from the University of New Mexico, Albuquerque, in 1963 and 1968, respectively.

He joined Sandia National Laboratories, Albuquerque, in 1961, as a staff member where he was engaged in many phases of nuclear and radiation effects studies, including the study of radiation effects on semiconductor devices and analysis of the vulnerability of electrical systems to a radiation environment. In 1973 he was assigned supervisory responsibility for the Computer-Aided Design and Analysis Division, and for developing computer techniques for use in the design, analysis, and layout of the large scale integration (LSI) circuits. Currently he is department manager for the IC Design Department and has responsibility for metal-oxide semiconductor (MOS) and bipolar IC design and testing, development of computer aids, and development and procurement of specialty semiconductor devices.

911461
2 of 2

An Overview of Logic Synthesis Systems

VINCENT P. TRAN
EXAMINER
ART UNIT 2

Louise Trevillyan

IBM Thomas J. Watson Research Center
Yorktown Heights, New York, 10598

Abstract

The term *logic synthesis* is used to describe systems that range from relatively simple mapping schemes to tools with sophisticated logic optimizations. In this tutorial, the requirements on logic synthesis systems will be discussed and the advantages and disadvantages of different approaches to logic synthesis will be presented.

Introduction

The goal of a logic synthesis system is to increase the productivity of logic designers by automating some or all of the logic design process. In general, this means that such a system automates the production of a technology-specific implementation of a logic function from some more abstract representation of the function. Proposals as to how to accomplish this goal range from straightforward, low-risk mapping schemes to very ambitious, but high-risk, methods of generating logic from high-level algorithms.

In a mapping approach, the design is entered at a low level, either in terms of technology primitives or in terms of "generic" functions which bear a close relationship to the target technology. Very few decisions about the form of the logic need to be made by the system. This method automates some of the more tedious aspects of logic design, and is low risk because the human designer has almost complete control of the resulting implementation, making the success rate for synthesis virtually 100%.

At the other end of the spectrum, there are systems that accept a logic specification in a form similar to a programming language. Almost all of the structural decisions about the logic are made by the synthesis programs. Insertion of memory elements, state assignment, and sharing functional units (ALUs, etc.) are examples of the issues that must be addressed. From the algorithmic description along with constraints on logic size and speed, these systems are expected to produce a complete, usable logic implementation. With current software technology, the likelihood of this being achievable on a system of any size is fairly remote, so the risk of relying on such an approach in a production environment is quite high.

There are also a large number of systems that inhabit the middle ground between these extremes. Most of these programs operate at the *register-transfer* level, in which the designer has made some decisions about the placement of cycle boundaries and the gross structure of the logic, but leaves the decisions about the local structure of the logic up to the program. As would be expected, the success rate of such systems falls between the success rates of the two extremes.

No matter how ambitious the approach, there are certain problems relating to logic quality and usability that must be faced by a synthesis system. Parts I and II of this tutorial describe these problems. In Part III, the approaches to synthesis are categorized. A single system in each category is discussed in order to illustrate how the problems of synthesis are addressed by the approach. The systems chosen are by no means the only such systems. They were chosen because there is enough published information about them to allow evaluation with respect to the criteria given in parts I and II. References [1,2,3] are recommended for additional descriptions and comparisons of recent synthesis systems.

I. Issues in Logic Synthesis

Many synthesis systems deal explicitly with the physical design of the implementation, but in this tutorial, only issues of logic design will be considered. The prerequisite for any synthesis system is that it reliably generate

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

RCL000114

© 1987 ACM 0738-100X/87/0600-0166$00.75

functionally correct, technology-legal implementations. Issues of logic quality are always secondary to those of correctness. Assuming correctness and legality, there are several other criteria by which the quality of synthesis-generated logic should be judged:

Area. This is the classical measure of the effectiveness of a synthesis tool and is usually the first problem addressed by any system. The goal is to make the area, whether measured in terms of cells of a gate-array, devices, equivalent-NAND, or square millimeters of a chip, comparable to that which can be achieved by manual design. Area is not the sole or final measure of the quality of the generated logic. Attention to other constraints, especially timing, can cause the area of the logic to increase. An implementation which meets timing constraints is superior to smaller logic which does not.

Speed. Logic implementations are subject to requirements on the minimum and maximum lengths of paths through the logic. These constraints can be stated in terms of actual arrival and required times expressed in units of time or they may be given in terms of levels of logic. In either case, it is desirable for synthesis to react to timing constraints by making intelligent space-time trade-offs. A good goal for a synthesis program is to generate the smallest logic that meets the constraints on path lengths. The issue of *power consumption* is related to both speed and area, and is usually part of timing correction. Using higher power will tend to speed up logic but will make the logic bigger and hotter. The decision as to the use of power must be balanced with time, space, and total power limit considerations.

Redundancy. A connection is *logically redundant* if it can be replaced by a constant without changing the logic function at any observable point (memory or output) in the design. It is necessary to remove redundant connections both because of chip testability requirements and because excessive connections lead to increased area, path lengths, and wireability problems. The source of redundancies can be the input specification or logic simplifications which cause connections to be moved or logic to be copied.

Physical Design. Although synthesis is not primarily concerned with physical design, some consideration of physical design issues is needed when performing logic design. Even if a synthesized implementation has met all of its logic-level constraints, it still is not a usable design unless it can be placed and wired, and, furthermore, it still satisfies the timing requirements after physical design has been performed.

Additionally, other topics such as error detection/fault isolation, self-test, and simultaneous switching could be addressed by a synthesis system.

II. System Characteristics

The other category of requirements usually has very little to do with the logic design objectives of a synthesis system, but is equally important in terms of a usable tool. No matter how good the quality of the logic, a synthesis system is useless as a design automation tool unless a logic designer can effectively and efficiently use it to help him do his job.

Capacity/Performance. The main justification for a synthesis system is usually that, with the advent of VLSI, the complexity and amounts of logic that need to be designed have increased so greatly that manual design is ineffective or too expensive. A useful synthesis system should have algorithms that scale in a reasonable way with the size of the design, and should either be able to handle very large flat designs or should provide hierarchical capabilities. If hierarchy is used, it is desirable to have some mechanism for cross-partition simplifications to handle global issues (such as redundancy removal).

Stability. We will call a system stable if a "small" change in the input specification produces a correspondingly small change in the output implementation. This is a useful attribute in cases where the designer is attempting to "tune" the design. For example, a designer loses productivity when a change to the specification to improve the timing on one path causes another path to become too long or an explosion in area in some other part of the logic.

Flexibility. The system should be able to accommodate varying input formats, technologies, design styles, and project requirements.

Output interfaces. The outputs of the synthesis system should not require manual manipulation to prepare it for further processing, and its output should be complete in the sense that everything needed for further processing is provided by the synthesis system. In addition to the design itself, this might include separate files of timing or verification data, graphic forms of the logic, or a form suitable for entry into a logic editor.

Interactive capabilities. It may be useful to allow the designer to "guide" the synthesis process by applying simplifications to only some parts of the logic or by varying the order of optimizations. One of the arguments in favor of synthesis is that the output is "correct by construction". If interactive systems include the capability of editing the logic, this advantage may be lost.

RCL000115

Usability. The system should be convenient for a logic designer to use. This means that consideration should be given to topics such as the design of the input language or editor, the amount of training a designer needs to use the system, and the understandability of the output of the system. Judging usability requires "hands on" use of a system. Since this was not possible, no attempt will be made to evaluate the usability of the systems discussed in this tutorial.

Other important topics related to synthesis are *logic comparison* and *incremental processing*. The goal of logic comparison is to verify that two logic models perform the same function. It can be used as a check on a synthesis system, when manual changes are made to synthesized logic, or in doing incremental processing. Incremental processing is most useful when employed together with an incremental physical design system. Given the large size of today's chips, physical design can be very time-consuming. When the logic is modified after physical design has been performed, it is useful if the synthesis system can bound the logic changes so that incremental physical design can be done. This can be done either by comparison of the original synthesized design with the new one and generating a change file, or by using hierarchical techniques in synthesis that bound the change to the size of the piece that has been modified. In the latter case, the responsibility is on the designer to provide suitably sized partitions in the hierarchy.

III. Approaches to the Problem

As has been mentioned, the range of synthesis systems, in terms of the problems they address, the sophistication of the algorithms, and the level of ambition of the systems, is very broad. In this section, we will examine five existing synthesis systems that typify the various approaches.

Polaris, a mapping system

One of the most simple, but extremely effective, synthesis systems is the one developed by Hitachi and used on virtually all of the chips in its M-68X processor [4,5]. The objective of this system is to always give the designer exactly the logic he expects with respect to gate counts and path lengths and to satisfy the electrical and physical constraints of the technology. The design is entered at a fairly low level in the ALDL language and the synthesis algorithm "maps functional specifications to target technology components on a one-to-one basis" [5 p 367]. The system does in fact make changes to the logic. It coalesces identical nodes and performs fan-in adjustment while translating the language into a graph form. It follows this by mapping the logical gate structure into allowed physical gates using a levelized polarity-propagation scheme. The system contains a table of physical gates that is ordered by desirability of use. The polarity propagation proceeds from the logic outputs toward the inputs, choosing the highest priority physical gate that includes the correct logic function and satisfies physical requirements (such as fan-in and dotting rules). The choice of the function to be used at a gate determines the required polarities of the logic that feeds the gate. Inverters are inserted when there is a polarity mismatch. Finally, a table-driven process is used to map the logic from physical gates into the physical units which correspond to the primitives of the gate-array.

In addition to synthesizing logic, the system is able to use complex modules specified by the designer and customize them to their place of use by deleting unused logic terminals. It can also copy hand-designed "basic modules" into the logic.

The Hitachi system addresses problems of correctness and legality but leaves issues of area, speed, etc. largely in the hands of the logic designer. The algorithms used appear to be linear in the design size, and, because it makes no dramatic changes to the logic, it rates high on stability and flexibility. It is not clear how many gates the system can actually handle, but the functional units used for synthesis appear to be in the 30-50 gate range. Logic comparison and incremental processing capabilities are available [6]. The small size of the models probably has more to do with the incremental synthesis and physical design capabilities than with any restriction on the synthesis system itself.

Productivity is gained in the Hitachi framework by relieving the designer of the necessity of doing many of the tedious and mechanical jobs required in gate-array/standard-cell design. However, most of the detailed logic decisions are still left to the designer to handle. Other systems which also operate at a register-transfer level are more ambitious in this respect. The large number of "higher-level" register-transfer level synthesis schemes can be classified roughly into three paradigms: algebraic, compiler-like, and expert-system. In the following sections, we will focus on an example of each of these approaches.

APLAS, an algebraic approach

Algebraic methods for automated synthesis generally are based on the approach pioneered by Quine [7], McCluskey [8], and others the 1950s and are rely on the minimization of *programmable logic arrays*, two-level AND-OR representations of the logic. (While most of the algebraic systems are based on PLAs, there are some that have taken a different approach, most nota-

RCL000116

bly the MACDAS system [9].) Much of the work done in this area has to do with *PLA minimization*, the goal of which is to reduce the number of AND terms in the implementation. There are known algorithms which will always lead to a minimal implementation (in terms of unrestricted AND and OR gates). These have very large computational requirements so that a medium-sized PLA with about 20 inputs is the maximum unit of logic that can be processed using these methods. The idea behind recent work has been to find new approaches, often employing heuristics, that will achieve or approach minimal solutions but avoid the long run times associated with the classical methods.

The APLAS system [10], developed at Stanford University, is an example of a PLA-based algebraic synthesis approach. The goal of the system is to develop a practical synthesis method that will implement the control logic portion of a design as PLAs.

The input to APLAS is a register-transfer level description done in DDL-P. The language is translated and produces a list of Boolean equations in terms of the register and logic inputs and outputs. The next process reads in the equations, converts them to an AND-OR form, and uses PLA techniques to to minimize the number of product terms needed to implement the function. In this case, heuristics are used in the minimization procedure, so an optimal solution is not guaranteed. In practice, the solutions are very good, and, for small examples, optimality is achieved.

The minimized form is further reduced by a procedure to optimize the number of connections in the PLA, and then the large PLA is partitioned into smaller PLAs in order to reduce area and improve speed. The partitioner also contains a redundancy removal algorithm.

This system addresses the major issues of synthesis. Since one of the advantages of PLA design is that it is very each to place and wire PLAs, this system also has good characteristics with respect to physical design. Programs that handle a design as a single PLA are notoriously poor with respect to capacity/performance, and this one is no exception (the maximum model reported has 72 inputs). The only design style supported by APLAS is PLA output and it would be difficult to change to a different design style. This is an inherent problem with this approach, but algebraic methods can be combined with other types of systems in ways the enhance their flexibility, as will be described later.

LSS, a compiler-like system

Another approach to the problem of synthesis has its roots in the observation that optimizing logic is similar to optimizing programs. The compiler-like synthesis systems tend therefore to draw on the work done in optimizing programming-language compilers. Compiler optimizations, such as common subexpression elimination, constant propagation, code motion, dead code elimination, and procedure integration, are re-cast into forms that are useful in logic optimizations. For example, in programming-language compilers, code motion might be used to move computations out of loops; in synthesis systems, the same techniques might be used to move late signals closer to the registers and outputs of the logic.

The synthesis system from IBM, LSS [11], is an example of a system built using compiler technology. IBM uses many different register-transfer level languages, and most of these can be used as input to LSS. The input is translated to a dataflow graph in which the nodes represent operations specified in the input (DECODEs, parities, adders, etc. as well as ANDs, ORs, and NOTs). The graph edges represent the data relationships in the logic. Peephole optimizations in the form of local, pattern matching transformations are applied at this level. The logic is then translated to a NAND or NOR level where global transformations similar to dataflow-based algorithms in programming-language compilers are used to simplify the logic [12] and redundancy removal processing [13] is applied. The early optimizations are largely directed at area minimization, with timing a secondary consideration. In the final phases, the logic is mapped into the target technology and detail timing analysis and correction are done. At this level, the focus is timing and power consumption with area being a secondary factor. These are generally accomplished by a global-analysis/local-correction approach.

Specific technology information in the form of tables and "user exits" is used throughout the synthesis process to aid in making optimization decisions. Externalizing such data allows a trained person to easily adapt LSS to new technologies and user requirements. Because global algorithms are being used, there is no guarantee of stable output - a small change to the source could potentially change a large window of logic.

LSS contains simplifications intended to address the major issues of logic synthesis. Like programming-language compilers, it is a "turnkey" system and the user interacts with it only at the source-code level. With the exception of full redundancy removal, the algorithms scale in a roughly linear fashion with the model size. (Complete redundancy removal is optional and used only when there are severe testability problems with the logic.) Models containing 40,000 two-way NAND equivalents have been processed using LSS. Logic comparison is done by using the SAS system [14]. LSS is used widely within IBM and hundreds of production bipolar and FET chips have been produced.

SOCRATES, an expert system

The major observation behind the expert-system approach to synthesis is that the task of synthesis, with its multiple, conflicting objectives, is a very difficult problem to solve algorithmically. The expert-system approach overcomes this problem by encoding the expertise of logic designers into a set of rules and allowing a system to apply the rules in varying sequences in such a way as to drive down the cost of the logic implementation. The significance of the arbitrary order of the application of the rules is that the search space of implementation possibilities is increased and the local minima typical of other approaches are avoided.

The SOCRATES system [15,16,17] General Electric Microelectronics Center and the University of Colorado at Boulder is a particularly interesting example of this kind of system because it illustrates not only an expert system, but a way in which an algebraic approach can be used in conjunction with another method to produce good technology-dependent, multi-level implementations.

SOCRATES can accept input in the form of Boolean equations, "linked" single-output PLAs, or a net-list. No matter what the input form, the logic is translated into a directed acyclic graph in which each node represents a two-level Boolean equation. Minimization is carried out at this level by using the ESPRESSO IIC logic minimizer [18]. This simplification is followed by another algebraic synthesis step which includes *weak division* (a form of factoring), and *multi-level minimization* (to take advantage of don't-care states). At this point, the algebraic portion of the system has designed the best technology-independent implementation it can. A library-mapping phase is then run to translate the function into a circuit in terms of a small number of generic primitives and the rules-driven portion of the system is run to complete the optimization.

The rules-based approach is used to take advantage of special features of the target technology. Each rule describes a local transformation that would, if applied, change a small window of logic so as to reduce the cost of the circuit in terms of area or performance. The tasks that the system performs are: (1) determine the set of rules which apply to the circuit, (2) compute the cost function for each application to determine the goodness of the moves, (3) select the move to be made, and (4) apply the selected rule to the logic and update the cost function.

The quality of the resultant implementation depends on the rules and the order in which they are applied. Because the rules interact, it is desirable to be able to look ahead through a series of rule applications when selecting the rules to apply. The rule ordering can be viewed as a tree in which the breadth represents the number of rules that can apply at a given place and the depth represents the sequence of transformations that can be done. In the ideal case, the complete tree of rule applications would be searched to find the optimal implementation (with respect to the rules). In practice, this is too expensive so the search must be limited to a certain breadth and depth of the tree. In SOCRATES, there is a second rule-based system, called the *meta-system*, that controls the search strategy and dynamically adjusts the scope of search in the tree.

SOCRATES addresses the problems of area and speed by accounting for these things in its cost function. It is less clear how the problem of redundancy is solved, for even if a completely testable design could be produced from the algebraic portion of the system, subsequent modifications to the logic cause logical redundancies to be introduced. Since redundancy is a global phenomenon, a local approach cannot deal effectively with the problem.

The flexibility of the system is enhanced by a "rule generation" module which allows a sophisticated user to enter and test a new rule in about three minutes. The ability to easily add such rules means that the system can be readily customized to new design styles, technologies, or project requirements. Again because of global algebraic algorithms, there is no guarantee of stability in this system.

Performance is a problem for most expert systems. In SOCRATES, even though great care has been taken in the system design and in efficient searching strategies, the results given in [15] do not indicate that the run time scales well with the size of the final design. The largest model the authors have reported running is less than 400 two-way NAND equivalents, but there is no reason to believe that this is anywhere near the maximum capacity of the system.

Flamel, a high-level system

The approaches discussed above all use register-transfer level input. The final area of synthesis attempts to raise the level of specification to an algorithmic level, so that the input would look more like a PASCAL or an ALGOL program. While such systems still fall into taxonomy given above, they are extended to make major structural decisions about the form of the logic. The system that is discussed here is compiler-like, but expert systems are also being used, most notably at Carnegie-Mellon University [19].

Flamel [20], developed at Stanford University, uses as input a program written in a slightly restricted version of PASCAL and execution frequency counts from a typical run of the program. Its goal is to find a hardware design with the same "external behavior" as the program which minimizes the estimated run time on the

RCL000118

typical data and which honors users' constraints on the cost of the implementation.

The program is translated into a control-flow graph in which the nodes represent the *basic blocks* of the program and the edges represent the flow-of-control of the program. Each basic block (a single-entry-single-exit section of code from the program) is represented by a dataflow graph together with some sequencing information. The program applies transformations to the control-flow graph to attempt to merge basic blocks in ways that can be used to enhance parallelism and reduce time. The choice of transformations to be applied is controlled by generating a tree of transformation combinations and picking the set of nodes in the tree that minimize the estimated time while adhering to a user-supplied constraint on resources (number of ALUs, registers, etc.). The time estimate is based on the number of levels in the block and on the frequency of use of the block as given in the execution frequency input. This process is followed by one that shortens the distance along critical paths by applying a level compression algorithm. Level compression also yields further opportunity for parallelism. Finally, Flamel optimizes resources by "folding" together pairs of resources that perform (or could be made to perform) the same function and are not used simultaneously. In some cases, the schedule will be lengthened to form more opportunities for folding in order to meet resource constraints.

Flamel's output is a description of the data path consisting of the ordered functional units (ALUs, adders, I/O pads, etc.) and the busses and their attachments, and a description of the control logic in terms of the states and transitions of a finite-state machine. All of the structural decisions have been made to get a design that meets the user's constraints. At this point, other tools, such as the register-transfer level systems described above, could be used to finish the task of detail logic design.

The results given for Flamel indicate that it is very responsive to the constraints supplied by the user and that it can quickly generate implementations from the high-level description. At this point, the resulting implementations are much larger than can be done manually; on the other hand, it can produce acceptable designs in a small fraction of the time required to do a manual design.

Summary

In order to be successful at logic synthesis, a system must do more than simply minimize area. It must generate implementations that are acceptable in terms of other measures such as speed, testability, physical design, and power. Furthermore, if it is to be useful in a VLSI environment, it must pay attention to the system characteristics of capacity, performance, stability, flexibility, and usability.

There are four basic approaches to the synthesis: mapping, algebraic, compiler-like, and expert-system. Each of them has been shown in real implementations to be viable, and each method has strengths and weaknesses in terms of the quality of logic that is generated and in the characteristics of the approach.

Since the goal of logic synthesis is to increase the productivity of the human logic designer, an evaluation of the success of the method should take into account the risk associated with using the system as well as the amount the system can do for the designer. A mapping approach may not be ambitious but it is very safe and provides a great deal of automation. On the other end of the spectrum, the "synthesis from algorithms" approach is the goal that everyone would like to reach, but it is not really practical yet in a production environment.

Acknowledgements

I would like to thank Bill Joyner, Terri Nix, and Dan Ostapko for their suggestions and help in developing this paper.

References

[1] W.H. Joyner, Jr., "Logic Synthesis", *Proceedings of Comp-Euro '87*, Hamburg, Germany, 1987.

[2] A.R. Newton, "Techniques for Logic Synthesis", 1985.

[3] A.L. Sangiovanni-Vincentelli, "An Overview of Synthesis Systems", *IEEE Custom Integrated Circuits Conference*,Portland, Oregon, 1985, pp. 221-225.

[4] T. Shinsha, et. al., "POLARIS: Polarity Propagation Algorithms for Cominations Logic Synthesis", *Twenty-first Design Automation Conference*,Las Vegas, NV, 1984, pp. 221-225.

[5] Y. Tsuchiya, et. al., "Establishment of Higher Level Logic Design for Very Large Scale Computers", *Twenty-third Design Automation Conference*,Las Vegas, NV, 1986, pp. 366-371.

[6] T. Shinsha, et. al. "Incremental Logic Synthesis Through Gate Logic Structure Identification", *Twenty-third Design Automation Conference*,Las Vegas, NV, 1986, pp. 366-371.

RCL000119

[7] W.V. Quine, "The Problem of Simplifying Truth Functions", *Am. Math. Monthly*, Fall, 1952.

[8] E.J. McCluskey, Jr., "Minimization of Boolean Functions", *Bell System Tech. Jour.*, Vol. 35, No. 6, November, 1956.

[9] T. Sasao, "MACDAS: Multi-level AND-OR circuit synthesis using two-variable function generators", *Twenty-third Design Automation Conference*, Las Vegas, NV, 1986, pp. 86-93.

[10] S. Kang and W.M. vanCleemput, "Automatic PLA Synthesis from a DDL-P Description", *Eighteenth Design Automation Conference*, 1981, pp. 391-397.

[11] W.H. Joyner, et.al., "Technology Adaptation in Logic Synthesis", *Twenty-third Design Automation Conference*, Las Vegas, NV, 1986, pp. 94-100.

[12] L. Trevillyan, W.H. Joyner, C.L. Berman, "Global Flow Analysis in Automatic Logic Design", *IEEE Transactions on Computers*, vol. C-35, no. 1, January, 1986.

[13] D. Brand, "Redundancy and Don't Cares in Logic Synthesis", *IEEE Transactions on Computers*, vol. CAD-5, no. 10, October, 1983, pp. 947-952.

[14] G. L. Smith, R.J. Bahnsen, H. Halliwell, "Boolean Comparison of Hardware and Flowcharts", *IBM Journal of Research and Development*, vol. 26, no. 1, January 1982, pp. 106-116.

[15] A.J. deGeus and W. Cohen, "A Rule-Based System for Optimizing Combinational Logic", *IEEE Design and Test of Computers*, vol. 2, no. 4, August, 1986, pp. 22-32.

[16] K. Bartlett, W. Cohen, A. deGeus, G. Hachtel, "Synthesis and Optimization of Multilevel Logic under Timing Constraints", *IEEE Transactions on Computer Aided Design*, vol. CAD-5, no. 4, October, 1986, pp. 582-596.

[17] W. Cohen, K. Bartlett, A.J. DeGeus, "Impact of metarules in a rule based expert system for gate level optimizations", *Proc. IEEE Int. Symp. on Circuits and Systems*, June 1985, pp 873-876.

[18] R.K. Brayton, et. al., *Logic Minimization Algorithms for VLSI Synthesis*, October, 1986, pp. 582-596. Hingham, MA: Kluwer Academic Publishers, 1984.

[19] T.J. Kowalski, D.J. Geiger, W.H. Wolf, W. Fichtner, "The VLSI design automation assistant: From algorithms to silicon", *IEEE Design and Test of Computers*, vol. 2, no. 4, August, 1986, pp. 33-43.

[20] H. Trickey, "Flamel: A High-Level Hardware Compiler", *IEEE Transactions on Computer Aided Design*, vol. CAD-6, no. 2, March, 1987, pp. 259-269.

TO SEPARATE, HOLD TOP AND BOTTOM EDGES, SNAP-APART AND DISCARD CARBON

2 OF 2

FORM PTO-892 (REV. 3-78) U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

NOTICE OF REFERENCES CITED

SERIAL NO.	GROUP ART UNIT	ATTACHMENT TO PAPER NUMBER
07/143,821	234	3

APPLICANT(S): Kobayashi et al.

U.S. PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
	A						
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

FOREIGN PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	COUNTRY	NAME	CLASS	SUB-CLASS	PERTINENT SHTS. DWG.	PERTINENT PP. SPEC.
	L								
	M								
	N								
	O								
	P								
	Q								

OTHER REFERENCES (Including Author, Title, Date, Pertinent Pages, Etc.)

R "Methods Used in an Automatic Logic Design Generator" by T. D. Friedman et al. IEEE Trans. on Computers, Vol. C-18, No. 7, July 1969, PP 593 - 613.

O "Experiments in Logic Synthesis" by J. A. Darringer, IEEE ICCC, 1980.

O "A Front End Graphic Interface to First Silicon Compiler" by J.H. Nash, EDA 84, March 198[illegible]

T "Quality of Designs from An Automatic Logic Generator" by T. D. Friedman et al. IEEE 7th DA Conference, 1970, PP 71 - 89.

U "A New Look at Logic Synthesis" by J. A. Darringer et al., IEEE 17th D.A. Conference 1980, PP. 543 - 548

EXAMINER: V. TRAN | DATE: 11/18/88

RCL000121

* A copy of this reference is not being furnished with this office action. (See Manual of Patent Examining Procedure, section 707.05 (a).)

IEEE TRANSACTIONS ON COMPUTERS, VOL. C-18, NO. 7, JULY 1969

Methods Used in an Automatic Logic Design Generator (ALERT)

VINCENT N. TRANS
EXAMINER
ART UNIT 234

THEODORE D. FRIEDMAN, ASSOCIATE MEMBER, IEEE, AND SIH-CHIN YANG

Abstract—The ALERT system converts preliminary high-level descriptions of computers into logic. The input to ALERT depicts the architecture of a proposed machine in a form of Iverson notation. As output, the architecture is "compiled" into Boolean equations, which may then be converted into standard computer circuits.

The purpose is to relieve designers of uncreative detail work. Complex structures may be represented conveniently by macro functions, algorithms, multidimensional arrays, and subscripted expressions. Control logic, intermediate registers, and selection and switching mechanisms are automatically supplied and the resulting design is consolidated and simplified by a variety of techniques. Methods used and reasons for those approaches are discussed. To elucidate operation of the system, a sample design is followed through its gradual development as it is processed by the successive steps of ALERT. Nine pages of circuit diagrams were automatically generated from the example and are included in the Appendix.

Index Terms—Architecture, automatic logic generation, compiler of computers, design automation, high-level computer description, Iverson notation, structural implementation of algorithms.

DEVELOPMENT of new computers at IBM has to a large extent been automated. The final stages of production are most fully mechanized, including wiring, circuit module identification, documentation, and control of manufacturing equipment [1]. Earlier development work such as checking, simulation, circuit selection, partitioning, and placement has been automated but to a more limited extent [9]. However, the initial task of logic design must still be accomplished by hand before the automated systems can be used. This work, which involves preparation of detailed diagrams or equations to specify the internal structure of a computer, is time-consuming, expensive, and frequently repetitious and uncreative.

The purpose of the ALERT system is to automate much of this preliminary logic design process. ALERT is a logic generator programmed to run on the IBM 7094. It inputs a description of a proposed computer expressed in high-level language. As output, it "compiles" logic designs which carry out the functions specified.

The input to ALERT, called the architecture, describes the overall computer system characteristics, and from this ALERT generates a logic design in the form of Boolean equations. These equations may then be processed by the regular IBM logic automation [9] and design automation [1] programs to produce circuitry and documentation for the desired computer. ALERT is analogous to a conventional compiler program because it translates its input from a high-level algorithmic form into a detailed machine-oriented form. But rather than generating a program as its output, ALERT generates logic designs for building hardware.

The nature and quality of the designs produced by ALERT is discussed elsewhere [5]. In this paper, the programs comprising the ALERT system will be considered.

Fig. 1 lists successive stages in the development of a new computer and identifies automated systems at IBM associated with each activity. With the aid of the ALERT system, the designer may apply automated techniques at an earlier point of the development than is now the case. In conjunction with other automation systems, development may then proceed from the initial stages through to final manufacture on a machine-aided basis. This should reduce costs, decrease development time, and aid checking and documentation.

The problem of automating this preliminary design work involves conversion of a high-level representation of a computer into a more detailed form. Before discussing techniques used, the choice of a high-level design language should be considered.

The High-Level Language

Traditionally, designers use informal notes, sketches, block diagrams, or natural language such as English to specify a computer before its logic has been designed. After the logic design has been prepared, the design itself may serve as the basic source document, but until the logic has been completed, a formal description is usually not available. A formal language for specifying computers at a high level would not only provide a possible starting point for automated processing, but it would provide an unambiguous source document for the various groups associated with computer development, such as architects, designers, engineers, marketing specialists, and programmers.

A high-level language should be clear, convenient, capable of specifying any feature of a computer, extendable, machine-readable, and it should allow the user to suppress irrelevant or repetitive detail.

Several formal languages have been proposed for the specification of computers at a high level. Among them are Schorr's Register Transfer Language [11], Schlaeppi's LOTIS language [10], a FORTRAN dialect by Metze and Seshu [8], and ALGOL dialects by Gorman and Anderson [6], and by Chu [2]. Each of these languages

Manuscript received October 4, 1968.
The authors are with the IBM Watson Research Center, Yorktown Heights, N. Y.

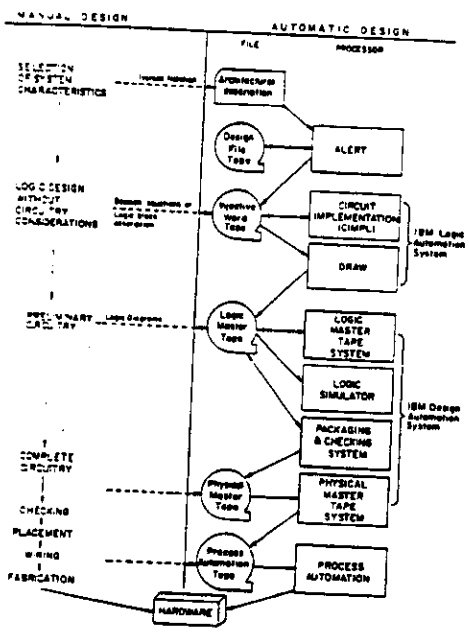


Fig. 1. Successive stages of development of a new computer and associated automation systems at IBM.

provides a means to formalize the information needed by architectural, engineering, design, and programming groups during the design of a computer. These languages convey differing degrees of detail in the representation of a machine, although each provides considerable flexibility in the amount of detail that may be included. In the case of the ALERT project, a descriptive language was chosen which is based on Iverson notation [7] which offers exceptional flexibility, convenience, and conciseness without sacrificing desirable features of the languages previously mentioned. Iverson notation has been applied in several fields, especially in the APL interpreter [3], and it has been used extensively as a high-level computer description language [4]. In particular, Iverson notation is an algorithmic language, which makes it possible to represent complex sequential logic as programs of microevents.

Some familiarity with the conventions of Iverson notation is desirable for proper understanding of the discussion in this report. The reader is referred to the summary on pp. 196–203 of [4].

It was necessary to develop additional conventions to depict system structure and data flow information for this application. Although Iverson notation has previously been used to describe the programming and functional characteristics of computer systems, these programming descriptions have not depicted the particular mechanisms used in those systems. To apply Iverson notation as a design specification language, the most important new convention imposed was that variables in the descriptions are considered to represent physical devices such as flip-flops, gate output lines, or registers. Because such devices constitute permanent hardware, declaration statements are needed to fix categories and dimensions of variables. A variable is automatically recorded as a single bit at its first occurrence in a description unless declared otherwise.

In accordance with this constraint, several Iverson operators which serve to modify operand dimensions or rank, such as compression or expansion, are excluded.

The entire computer description is separated into microprograms which describe data transformations in the computer. Each microprogram is considered to possess individual timing and control logic so that it may execute concurrently with other microprograms.

In addition to the regular microprograms, special macro functions may be defined in computer descriptions. Such functions can be invoked by and included as subsections in other microprograms.

Declaration statements are distinguished by a *D* in column one of the punch card, the start of microprograms by an *M* (or *MM* if manual specification of control logic is desired), and macro functions are denoted by an *F*. In addition, new statement types were introduced, such as assignment statements which affect only set or reset lines of flip-flops. Optional symbolic statement labeling is permitted.

Because the input is punched on cards to be accepted by the 7094, symbols in Iverson notation must be transliterated as shown in Table I. An innovation in this version of Iverson notation is the period used as statement terminator: all entries to the right of a period in a card are considered to be comments.

A complete example of a computer description for ALERT processing may be found in [5].

Mechanization of the Logic Design Process

At this point, specific techniques will be considered for processing the input description to obtain an acceptable logic design.

Design by humans involves strategic planning and subsequent development to carry out the plans. The goal in the ALERT project was to delegate development work to machines whenever possible. This requires that strategic aspects of design must be distinguished from development.

Unfortunately, there is not always a clear distinction between these two aspects of design. For example, the choice of a computer instruction repertoire is clearly strategic and must involve the human. Connections of logical components in a binary tree decoder is generally more routine and would be appropriate to delegate to machines. But certain details of the control logic, for example, may be best determined by human judgment while other details fall into place in a mechanical way.

TABLE I

TRANSLITERATION OF APL SYMBOLS FOR KEYPUNCHING MACHINE DESCRIPTIONS FOR ALERT

Symbol	Meaning	Transliteration
$+ - \times \div$	Arithmetic operations	+ - * 'DIV'
$\star$	Exponentiation	**
$\wedge \vee \sim$	Logic operations	'AND' 'OR' 'NOT'
$\mid$	Residue, absolute value	'RESIDUE' 'ABS'
$< \leq > \geq$	Relations	'LS' 'LE' 'GR' 'GE'
$= \neq$		'EQ' 'NE'
α	Prefix	'PREFIX'
ω	Suffix	'SUFFIX'
,	Catenation	,
/	Reduction	/
$\perp$	Base 2 Value	'VALUE'
ϕ	Left, right shift	'LEFT' 'RIGHT'
$\leftarrow$	Assignment	=
$\rightarrow$	Branch	GO TO
$\rightarrow(X)/S$	Conditional branch	IF (X) GO TO S.
[ ]	Subscript	( )
;	Subscript delimiter	,
:	Label	.
Carriage return	End of statement	.

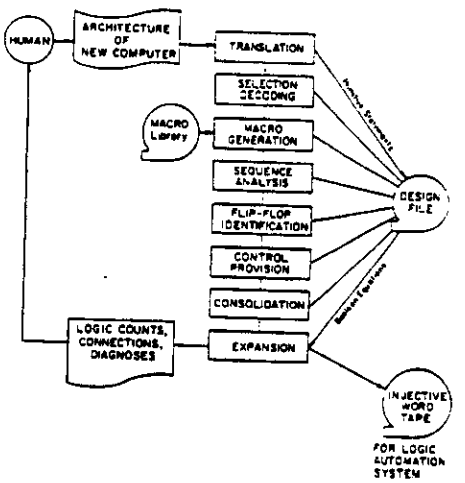


Fig. 2. Transformation steps of ALERT.

It is not desirable to impose a priori rules concerning what the user should and should not specify. Here the flexibility of Iverson notation proves valuable. Default options are provided so that the designer can work at the level of detail he desires. He may choose to omit details, and in those cases the system will refer to the default option to provide the details according to predefined conventions.

To mechanize the design process, a series of explicit transformation steps have been defined, each of which serves to bring the computer description closer to the final logic design. Each step consists of a set of subroutines concerned with some aspect of the design process. The steps systematically transform the machine description by supplying detail needed to complete the logic design.

The transformation steps are provided only as a convenience to the user so that he may avoid detailed work. Whenever the user wishes to specify details explicitly, he may override the system.

There is considerable latitude possible in the choice and order of these steps. In ALERT, the design process was separated into eight major steps called translation, selection decoding, macro generation, sequence analysis, flip-flop identification, control provision, consolidation, and expansion. (See Fig. 2.)

During the successive transformations from architecture to logic, the desired computer is represented internally by a single common format. The internal representation language is sufficiently general to depict a computer at the high architectural levels or down to a bit-by-bit account of the logic. The representation resides in the master design file, a threaded linked-list structure organized for rapid retrieval and manipulation of data.

To apply the ALERT system, just as in the traditional approach the user first selects overall features of the desired machine, and then he must formulate a machine organization to provide those features. The instruction set, major registers, and data flow paths should be identified, but routine details involving timing control, addressing logic, or hidden registers may be passed over and left for ALERT to work out. At this point the user is ready to prepare the architectural description.

The architectural description begins with declarations of the storage, channels, important registers, toggles, and other devices which the user wishes to note. Predesigned blocks of logic such as decoders or adders may be described as macro functions. Data flow paths are represented as register transfers using assignment statements. Memory access, indexing, and interrupt processing may be specified as microprograms. The instruction set can be indicated as a set of microprograms, and the overall control should be specified as a master microprogram. Such a document serves as a formal high-level description of the intended machine.

The user punches the architectural description onto cards and inputs it to the processor. The eight steps of the ALERT system then successively transform that description into logic.

In the following section, we shall examine the transformation steps in terms of their purposes and methods. To exemplify the processes, we shall show how each step helps to transform a typical Iverson statement from architecture to logic.

The sample statement taken from an architectural description written by A. Falkoff defines a microprogram algorithm to accomplish the load index instruction

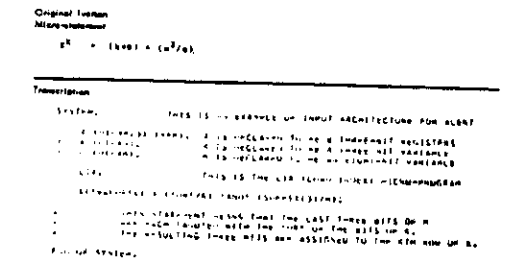


Fig. 3. A microstatement in Iverson notation and its transcription keypunched as a computer description.

in a small computer. It was originally specified in Iverson notation as

$$X^k \leftarrow (k \neq 0) \wedge (\omega^3/m).$$

In this statement, X represents a matrix of eight 3-bit index registers, m is an 8-bit register, and k is an integer variable between 0 and 7.

This statement indicates selection of row k of X, which in fact identifies index register number k. That register is loaded with the value of the last three bits of m, provided that k is not equal to zero. If k is zero, then the number k index register (i.e., index register number 0) is set to zero.

This statement appears deceptively simple, and thereby illustrates the conciseness of Iverson notation. It implies existence of gating to set or reset any of the eight 3-bit index register, i.e., 24 bits. It also implies existence of a mechanism to select the appropriate register as well as provisions to avoid data transfer when $k = 0$.

To transcribe this statement, k was changed from a scalar integer variable to a bit string (which will in fact be three bits long). For keypunching we represented this bit string as "K." The expression "$k \neq 0$" then was reformulated as "$(\perp K) \neq 0$," or more simply, as "$\vee / K$."

In order to elucidate the nature of the ALERT transformation steps, this statement was processed in isolation as though it were an entire machine description. A complete input deck was prepared describing an imaginary computer which does nothing except load its kth index register. This deck is listed in Fig. 3 with the names and Iverson operators transliterated for keypunching along with the required declaration and heading statements.

We will now examine the individual ALERT steps and their effects upon this machine description.

The Transformation Steps of ALERT

The first routine to process the input was the translator.

Translation

This routine scans the modified Iverson notation and converts it into a more constrained format used in the design file. This routine also checks the input for format, syntax, and consistency. The input language permits considerable syntactic complexity including indefinitely nested expressions, symbolic statement labeling (labels may be referenced with variably subscripted names), and context determined interpretation of operations (as described below in the section "Constants in a Description").

The internal design file representation is more restricted. The file is composed of fixed-format "primitive" statements, each of which may depict only a single type of gating function along with the signal sources and destinations. These primitives provide a framework upon which the desired logic design is built. The translation then is a parsing, analysis, and decomposition of the input statements into a simpler more design-oriented form. The procedure-oriented input reflects data flow and control transactions, while the design file more directly indicates componentry and paths implied by that flow. During translation, intermediate variables implied by Iverson expressions but not explicitly represented in the microprograms are generated and filed along with the declared elements.

Translation is a literal reformulation of the architecture, and the user must take care to express his input so as not to imply an obviously redundant structure. This need occurs in the example in Fig. 3. The expression "$(k \neq 0)$" in the original Iverson statement could be rewritten as "$((\perp K) \neq 0)$" when keypunched, but this would imply existence of a complicated mechanism to compare the value of the bits of K with a special vector whose bits have the value zero. The expression "$(\vee / K)$" is preferable because it implies only a simple OR gating which achieves the same effect as the more elaborate mechanism.

The translator also must assemble the statements into the proper design file encoding. As each variable name is scanned, it is relegated to a name list. The name's address in this list is not directly entered into the design file, but a threading is calculated for all references to that variable.

The design file is forward-threaded so that each reference to a variable name is actually a pointer to the address of the next occurrence of that name in the file. Only the last occurrence identifies the variable itself. Threading makes it possible to retrieve all references to any variable with a minimum number of file accesses. The entries of the design file are packed two to a 36-bit 7094 word. Four bits are used to designate type of entry and 14 bits constitute the entry itself. The purpose of packing is to double the number of entries in the file (but, of course, at the expense of additional processing needed to pack and unpack entries).

Experience using ALERT has indicated that threading successfully saves processing time, but that the data packing is of less critical value, especially since the design file must overflow onto tape for a machine design of substantial size.

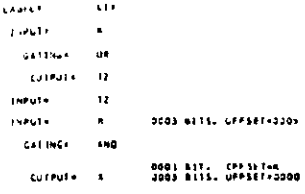


Fig. 4. Printout of the design file after the translation step.

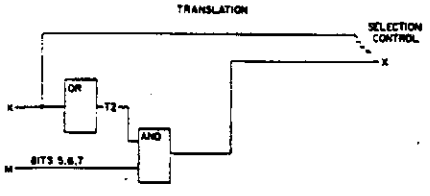


Fig. 5. Schematic diagram of the structure represented in the file after translation.

Fifteen types of entries are distinguished, including signal sources and destinations, subscripts (constants or variables), statement labels, microprogram headings, gating operators, patching markers, and even pointers to comments (comments may be recorded on a separate tape).

It may be instructive to examine the design file printout in Fig. 4 to observe the result of translating the sample machine description presented in Fig. 3.

Two design file statements were produced signifying two gating structures. The first statement (lines 2, 3, and 4 of the listing) indicates that the (three) components of register K are connected to a single OR gate. To represent the output of this gate, a new variable name, T2, was generated by the translator. This variable, which represents a single wire, is connected to a structure of AND gates according to the second design file statement. These gates serve to inhibit or pass three consecutive bits from M beginning with a bit five positions from the left end which corresponds to the expression "(SUFFIX(3)/M)." The outputs from these AND gates are specified as having for their destination all three bits of the Kth row of the array X.

The reader should not be discouraged by the intricacy of this explanation since the format is not organized for human reading. Details of the design file encoding such as identification of array components (the SIZE and OFFSET in Fig. 4) are properly beyond the scope of this report. In the following sections we will instead refer primarily to diagrams that were drawn by hand which depict the structures represented by the contents of the design file. Fig. 5 is such a diagram showing the structure represented in the file after translation. The reader may note a one-to-one correspondence of file entries and elements of the diagram. Note that, in this and successive diagrams, individual elements are not necessarily distinguished in arrays.

After the entire architectural description has been processed by the translation routines, another group of routines further transforms the description in the design file. These constitute the selection decoding step.

Selection Decoding

It is sometimes useful in a computer description to attach a variable subscript to an array name, as for instance in the expression X('VALUE' K) in Fig. 3. This is taken to mean that component elements of the array (X in this case) may be selectively addressed during operation of the proposed machine, and that they are addressed according to the value of the subscripting variable (K in this case). Variable subscripting provides a shorthand for representing complicated logic structures.

In ordinary program-generating compilers such as FORTRAN, variables represent values rather than structures, and variable subscripts are readily processed by generating code to add the value of the variable to the array address during execution. In a design compiler, variable subscripts present more difficulty. Selection of an element in an array requires sufficient logic to open the data paths to the desired element, but to no others, according to the contents of the selecting variable. Moreover, the selection logic for arrays used as signal destinations is quite different from that for arrays used as signal sources.

The selection decoding programs scan the design file and when a variable subscript is discovered, it is replaced by a block of logic to provide selection. In particular, the output of the subscript variable is fed to a decoder to produce a selection vector. Each bit of this selection vector is gated to control availability of one element of the array. Arrays of one, two, or three dimensions are permitted and up to three variable subscripts may be used to reference an array.

In the case of the sample design, the variable subscript in the term X('VALUE' K) signifies that components of the array X are selected according to the current value of the bits in K. Since this expression occurs on the left side of an assignment statement, the selection decoding routines recognize that the selection affects the input lines to X. This capability is provided by supplying a decoder tree for the bits from K. The output from this tree is used to control access to the X bits. Fig. 6 lists the contents of the design file after these automatic modifications have been made, and Fig. 7 is the manually prepared diagram of the design at this stage. The decoder is represented by a macro operator which is processed by the next step.

Macro Generation

In addition to the elementary logic operators, various higher order functions may be specified in the machine description. In the preceding transformations, such high-order operators were treated as black boxes. The

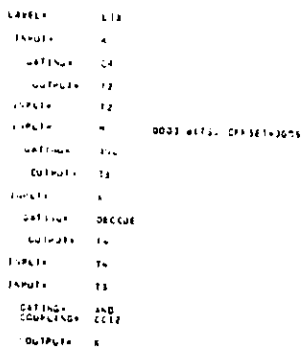


Fig. 6. Printout of the design file after selection decoding.

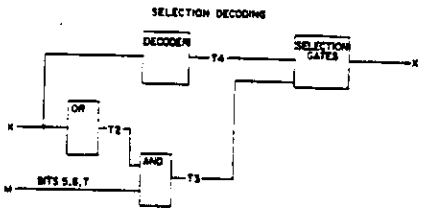


Fig. 7. Schematic diagram of the structure after selection decoding.

macro generator replaces these black boxes by the complete combinational logic required to accomplish those operations. This involves replacing any occurrence of a high-order operator in a microstatement by an entire block of elementary logic. Such a process is similar to generation of detailed coding by an ordinary assembler program when a macro is encountered; and the techniques used are analogous to those of conventional macro processors.

These blocks of logic may be copied literally from libraries of such blocks, or they may be custom-generated by logic-designing algorithms. In addition to the high-order functions specified by the designer in the original architecture, macros are supplied for decoders, counters, and shifters.

Decoders, for instance, are generated as simple binary trees. Other high-order operations, especially the arithmetic functions, may entail a number of special considerations. There are many alternative designs, for example, of adders. If the user does not specify his own macros, default macros are automatically provided.

In the sample design, a macro reference to a decoder was entered into the file during the previous step. During the current step, the design file was scanned by the macro generator and the operator "DECODE" was recognized as a macro name. Logic for a binary decoding tree was generated using the arguments K and T4 as the decoder's input and output, respectively. The routine automatically determined the size of the decoding tree

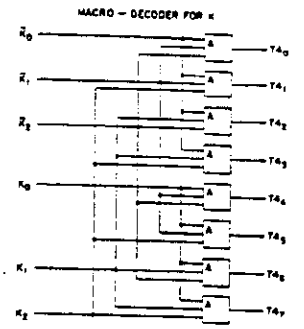


Fig. 8. Diagram of the decoder produced by the macro generator.

according to the number of bits of the arguments. The design produced by the macro processing is depicted in Fig. 8.

Sequence Analysis

During this step, control and sequencing requirements for the design are determined.

The user may choose to specify explicitly all control logic within sections of the design where timing or other constraints warrant particular attention to detail (as in memory accessing, for instance). Within less critical sections of the design, however, the user may omit routine details of the control. This approach is called *implicit* timing, and control is automatically derived according to the sequence inherent in the microprogram algorithms of the architectural description.

An algorithm specifies a discrete sequence of microevents. But it is not desirable to assign every microevent a distinct time period in a sequential network. Instead, the microevents are grouped together into as few separate time periods as is possible. This will allow concurrent parallel operations.

The sequence analysis routine partitions the microprogram into groups of operations which may occur during the same time period or event. In the case of those microprograms having manually specified control logic, this processing is not required. In the case of microprograms with implicit timing, the sequence analysis program must distinguish successive control periods. This is accomplished by scanning each microprogram in the design file from its beginning and determining points where new event groups must occur.

A microprogram is partitioned into event groups by the following algorithm.

1) The beginning of each microprogram is established as the partition point of an event group.

2) Statements which are destinations of GO TO's are established as partition points. (GO TO's signify deviations from sequential flow of microprocesses.)

3) Conditional test statements (IF's) are followed by partition points.

4) Each event group is scanned from its initial partition point, and whenever any variable previously encountered in that group is to receive a signal, then the statement under examination is established as a new partition point.

The partition points are recorded in a sequence of events table which is later used to provide control logic.

ALERT was programmed so that each implicit microprogram is provided with a single series of event table entries. It should be mentioned that as an alternative approach, a microprogram could be separated into subsections, each having distinct sequences. In this case, the total number of events could be reduced, but additional control would be needed to distinguish the subsections.

In the table, the event numbers may not follow a strictly one-by-one progression. Certain gating operations may continue for a period of several time units when they are performed, and the table entries must reflect this. Furthermore, the designer may set the next event count to a value relative to a previous statement in order to impose additional timing constraints.

When the sample design previously mentioned was processed, the sequence analysis determined that all microprocesses could be accomplished within a single time period. Just one partition point was established at the beginning of the microprogram. Two event periods, however, are required. The first period (event number 0) is an inactive phase required for every microprogram to indicate that it is not in execution. The other period (event number 1) is the single active phase.

Identification of Flip-Flops

In the previous steps, we did not distinguish between logic structures which produce output only momentarily while they receive input, and structures which must attain states which persist after their input ceases. The purpose of this routine is to identify variables which need to retain their values over some period of time.

The microprograms are scanned for any variable which serves as a signal source during a later event period than when it acquired its value. Such a variable must be able to hold its value, and accordingly it is identified as a flip-flop. A flip-flop (or latch) is a bistable circuit element which stores a bit of data until some new input alters its state. The value of this datum is available as the flip-flop's output.

In addition to this automatic search for flip-flops, the user may declare variables to be flip-flops by writing "(=FF)" following the variable's name (as in the case of X in Fig. 3). To avoid possible omissions, the user should declare any variable to be a flip-flop when he intends it to be so. Likewise, whenever he intends a name to represent a register, he should declare it to be a vector of flip-flops.

Every flip-flop is considered to have a SET input which puts the device into the "ONE" state, and a RESET input which puts it into the "ZERO" state. Two new types of assignment operators were provided to correspond to these inputs, namely "$=$'SET'" and "$=$'RESET'". A statement of the form,

$$A = \text{'SET'}\ B,$$

specifies that the output from bit B is connected to the SET input line of flip-flop A. Likewise,

$$A = \text{'RESET'}\ B,$$

specifies that the output from B is connected to the RESET input of A. For formal completeness, we may define these assignment operators as:

$$A = \text{'SET'}\ B \quad \text{means } A \leftarrow A \vee B;$$
$$A = \text{'RESET'}\ B \quad \text{means } A \leftarrow A \wedge \bar{B}.$$

A 0 or 1 signal delivered to the SET input of a flip-flop causes the device to acquire its previous state ORed with the value of the SET line. RESET input of value 0 causes the flip-flop to remain at its previous value, while a RESET input of value 1 assigns the value 0 to the flip-flop.

If a variable P in the design file is discovered to be a flip-flop, then any design file statement of the form

$$P = Q$$

will automatically be rewritten as two statements:

$$P = \text{'SET'}\ Q,$$
$$P = \text{'RESET'}\ \text{'NOT'}\ Q.$$

This specifies that Q and its complement $\bar{Q}$ are connected to the SET and RESET inputs of P, respectively. These two connections serve to jam the value of Q into flip-flop P.

Control signals for passing or blocking data to flip-flops must be distinguished from the data themselves. The control signals, like the data, must be duplicated and connected to the SET and RESET sides of flip-flops, but unlike data, control must not be complemented at the RESET side. This allows precisely the same flip-flops to be selected for SET and for RESET operations.

Referring to the sample design as developed in Fig. 7, the output lines from the selection gates are loaded into X. But X was declared to be a flip-flop array. The flip-flop identification step modifies this logic so that the positive values of these lines are directed to the SET input terminals of the X flip-flops. However, a second bank of gating is then automatically provided to supply complemented input data to the RESET terminals of X. (See Fig. 9.) Note, however, that NOT gates are used only to complement the data (the T3 lines), but that the selection vector, T4, which controls access to the appropriate components of X, is not complemented.

1) This routine searches for identical operators which have the same inputs. Such operators are respecified as one consolidated operator, and the original event control signals are used to enable each output at its appropriate time.

The original inputs and outputs are gated by their respective event control signals.

For example, suppose the following statements occur in the design file.

Statement no. [1]: A = B 'AND' C.
[2]: D = E 'OR' A.
[3]: F = D 'AND' S1(1).

[24]: G = B 'AND' C.
[25]: H = E 'OR' G.
[26]: J = H 'AND' S1(4).

Here S1(1) and S1(4) are timing control signals. The consolidation routine eliminates statement [24] and then G, wherever it occurs, is replaced by A. Statement [25] then becomes $H = E$ 'OR' A. This new statement is, in turn, eliminated and H is replaced by D. Statement [26] will finally become $J = D$ 'AND' S1(4).

This process is repeated until no identical gates are found in the file.

2) Another function of consolidation routine is to reduce gatings by rearranging the logic.

Until consolidation, arrays have been depicted as simple monolithic entities. Connections involving vectors or matrices were represented as though only single units were joined when in fact arrays of connections and gates were implied. Many more lines and gates may be required for the connections that were actually chosen than would be needed for logically equivalent connections.

The routine examines logically equivalent variations of the expressions and counts the components needed for each variation. The variation requiring fewest elements is selected.

3) If signals are assigned to a flip-flop in more than one micro-statement, the signal sources are ORed together so that only one assignment is provided. In this way, all signals used to SET or RESET any flip-flop are collected into common connections.

The result of consolidation on the sample design is shown in Fig. 11. Because of its small size, only the rearrangement part (item 2) of the consolidation step affects this design. The consolidation routine discovered that the gating arrangement in Fig. 10 is wasteful. The S1(1) signal is reconnected so that it is ANDed with T4 instead of T14 and T16 as in Fig. 10. The number of individual gates was reduced by this process from 96 to 56.

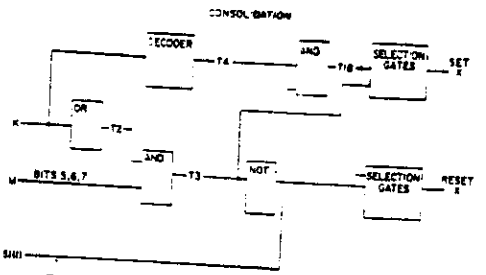


Fig. 11. Diagram of the structure after consolidation.

Expansion

By this point the logic design is nearly complete, and it only remains to output the results in the form of Boolean equations.

The design file entries, however, may actually represent arrays of connections. A single statement may signify a large number of individual connections, and these may be connected in complicated patterns.

The purpose of the expansion step is to report every device and connection individually. During this step, each statement in the design file is examined, and when variables are discovered to comprise arrays rather than single bits, separate copies of the original statements are generated for the separate components of the array. As the statements are recopied, component identifications are calculated according to control terms in the file. The expansion results in such a marked enlargement of the file that the threading conventions become inconvenient and are dropped at this point. An exhaustive connection-by-connection exposition of the logic is produced.

When the sample design was expanded, K was found to be a three-bit vector, M involved three bits, and 24 SET lines and 24 RESET lines were required for the eight three-bit index registers denoted by X. In addition, the intermediate system-generated variables T3, T4, and T18 were discovered to be arrays. Therefore, statements which involved these variables were repeatedly copied with components individually identified which resulted in 140 gating statements. Seventy-two of these statements represent "gates" in only a purely formal sense since they are actually a renaming of an element. (The renaming is due to techniques used in processing. It is not of concern since these "gates" are automatically eliminated later.)

A printout of the design at this point in Boolean equation format is shown in Appendix I. The first equation in this listing,

$$T2 = K/0 + K/1 + K/2,$$

signifies that the three bits of K, i.e., $K(0)$, $K(1)$, and

$K(2)$,[1] are connected to a common OR gate, the output of which is denoted T2. The next three statements indicate how this output is connected with the three bits from M to produce a three-bit variable denoted as T3. The next equations present an example of a "formal" intermediate variable and a "formal" gating function. This is actually only a renaming of a variable for bookkeeping purposes: the variable T5 is exactly the same as the variable K. The remainder of the printout may be interpreted similarly.

At this point, processing by the ALERT programs has ended. As mentioned earlier, other automation systems transform logic equations into circuitry, and these are now called in to complete the machine design which has been developed by the ALERT program.

Further Processing After ALERT

The IBM logic automation system accepts input in the form of Boolean equations as produced by the ALERT system. This system accomplishes a type of Boolean minimization and then converts the resulting logic into circuit gates of the IBM solid logic technology family. This entails reorganization of the logic to satisfy circuit engineering requirements. The logic automation system also lays out the resulting logic in engineering diagrams. Finally, the design is recorded into standard engineering files by the IBM design automation system and from these files the machine may be simulated, tested, and manufactured.

When the sample design in Appendix I was processed by the logic automation programs, it was discovered that 72 "gates" were actually only renamings of variables (as noted in the preceding section), and therefore these "gates" were eliminated. The program also discovered that the variable T2 is precisely the complement of another variable, T4(0), and accordingly T2 was eliminated. After the logic automation minimization step, 67 logic gates remained. These, however, are ideal logic gates which do not conform to the engineering constraints of solid logic technology circuitry. Only a limited choice of circuits in this technology were available to provide gating functions, and these did not match the ideal gates specified. The logic automation program automatically replaced the ideal gates in the design with available circuits and then proceeded to supply additional gates to correct for factors such as module input pin restrictions, fan-out limitations, and signal polarity inversions caused by the circuits. When all modifications were made, 191 gates resulted. These were automatically diagrammed by the logic automation system into nine pages of logic sheets, as shown in Appendix II.

In the first diagram, for example, lines from the bits of K are passed through inverters (denoted by N) and connected to an AND-inverter gate labeled "3C-CC." Notice a line drawn off this gate's output labeled "T4 0." This is the complement of the variable T4(0) which is one of the decoded signals formed from the bits of K, and it corresponds to the variable T2.

Although there has been considerable rearrangement of the logic during the course of the design, the reader may trace out in the nine diagrams all functions originally specified by the Iverson notation. The diagrams constitute an "implementation" of the architectural description in solid logic technology circuits. The circuitry was generated directly from the architecture untouched by human hands.

Conclusion

The sample architecture description in Fig. 3 was chosen to elucidate the operation of the ALERT system. Actual computer descriptions are, of course, far more extensive. However, the example illustrates the large quantity of detail that may be generated from a simple statement in Iverson notation. The machine description in the exact form shown in Fig. 3 was processed and the actual results of each processing step are depicted in Figs. 4 through 11 and Appendix I. Appendix II presents machine-drawn diagrams which were automatically derived from the final design produced.

ALERT thus is an operational logic generator. Several experimental techniques were introduced to assist logic design. Some techniques, such as selection decoding, appear to aid the designer only occasionally, while other processes, such as macro generation, seem widely useful and warrant further development.

The results suggest that a compiler of logic may provide the sort of improved usage of design manpower that programming compilers have provided to programmers. This implies faster development of new computers, immediate documentation, and also improvement of quality because the architect/designer may try alternative plans without waiting for manual logic design and choose the best from among these preliminary designs.

Appendix I

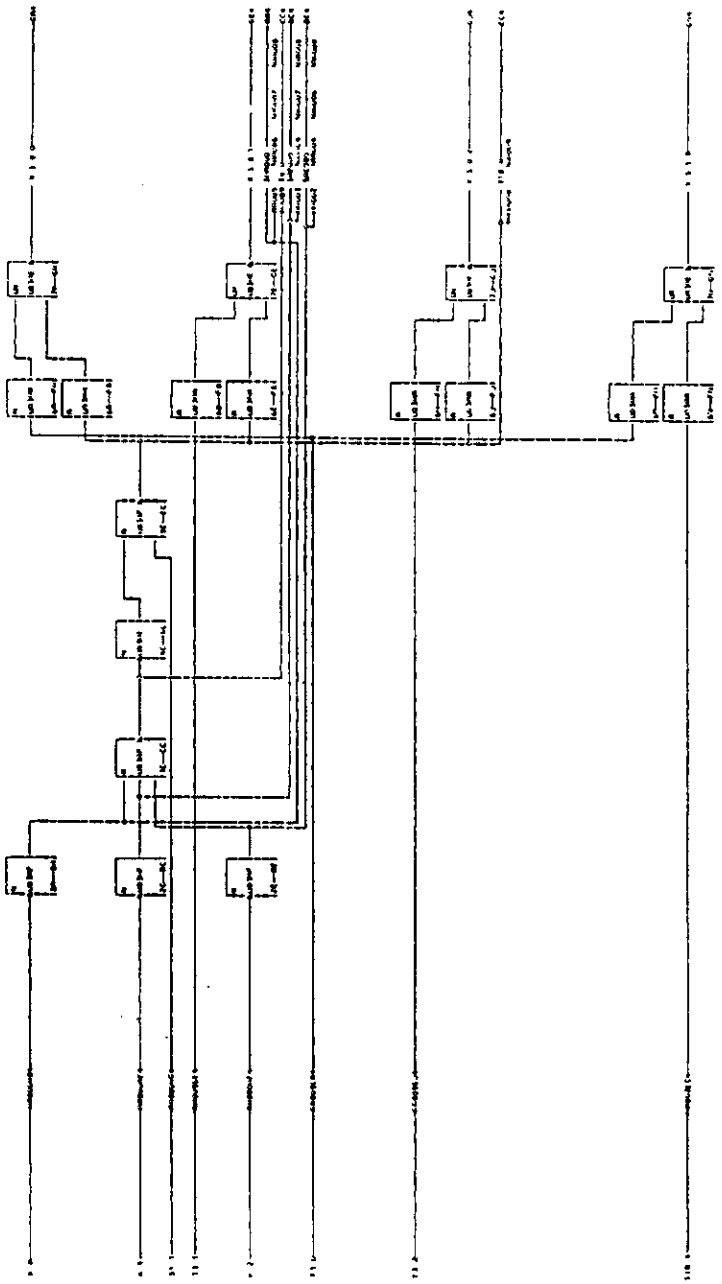
Printout of Boolean Equations Generated by the Expansion Step

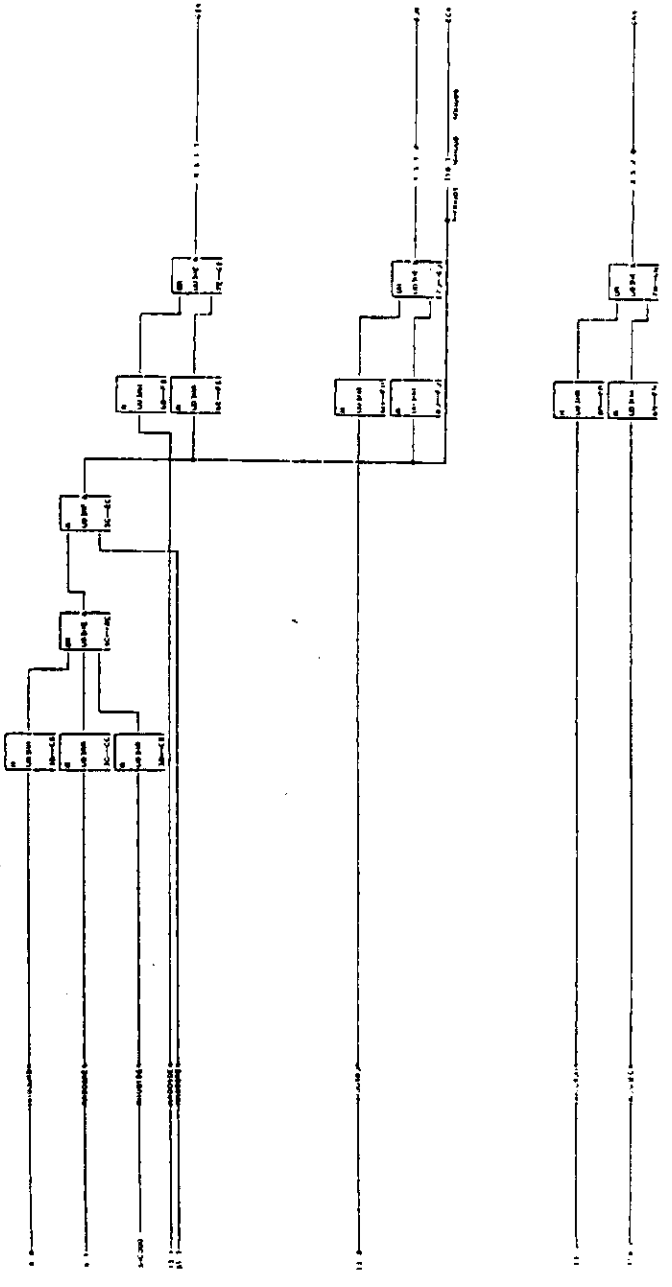
```
T2      =   K/0+ K/1+ K/2
T3/0    =   T2* M/5
T3/1    =   T2* M/6
T3/2    =   T2* M/7
T5/0    =   K/0
T5/1    =   K/1
T5/2    =   K/2
T4/7    =   T5/0* T5/1* T5/2
```

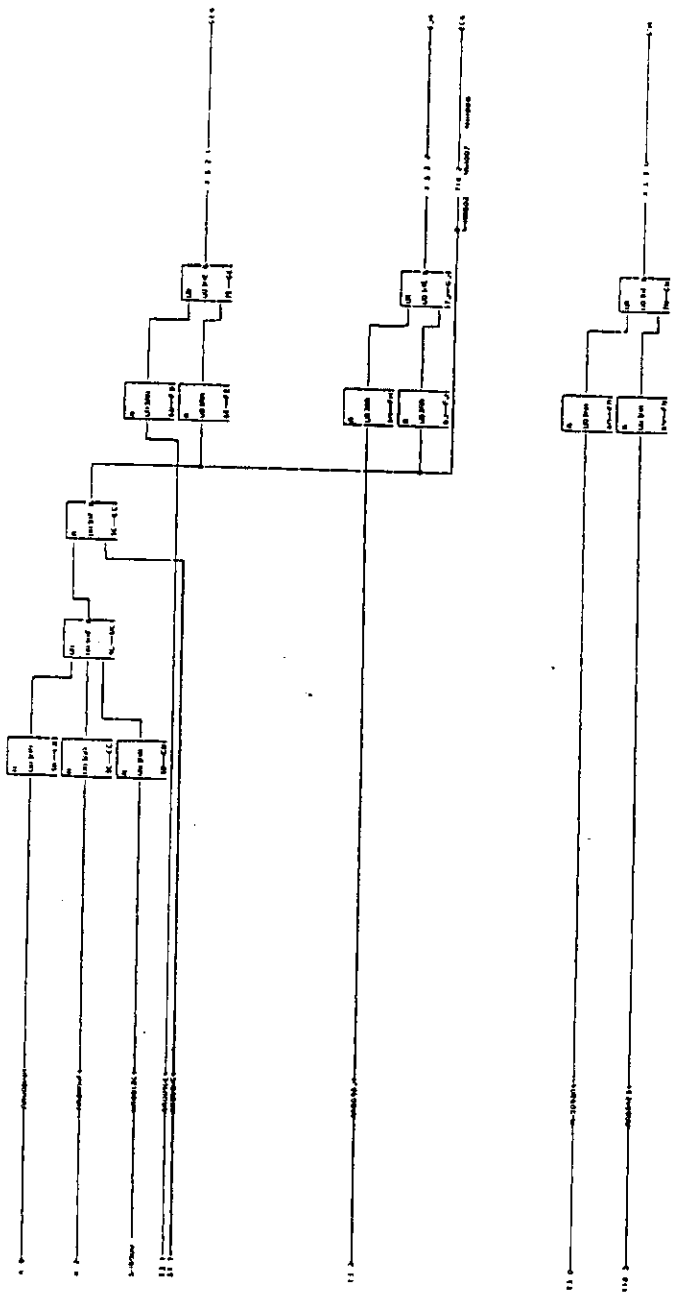
[1] Unfortunately, formatting conventions for arrays differ in ALERT, in the Boolean equation format, and in the engineering diagrams since the three systems were developed by separate groups. The variable "$K(0)$" in ALERT is represented as "$K/0$" in the logic equation format, and this same variable is printed out as "K0" in the engineering diagrams. Likewise, NOT in logic equation format becomes a minus sign, OR a plus sign, and AND an asterisk.

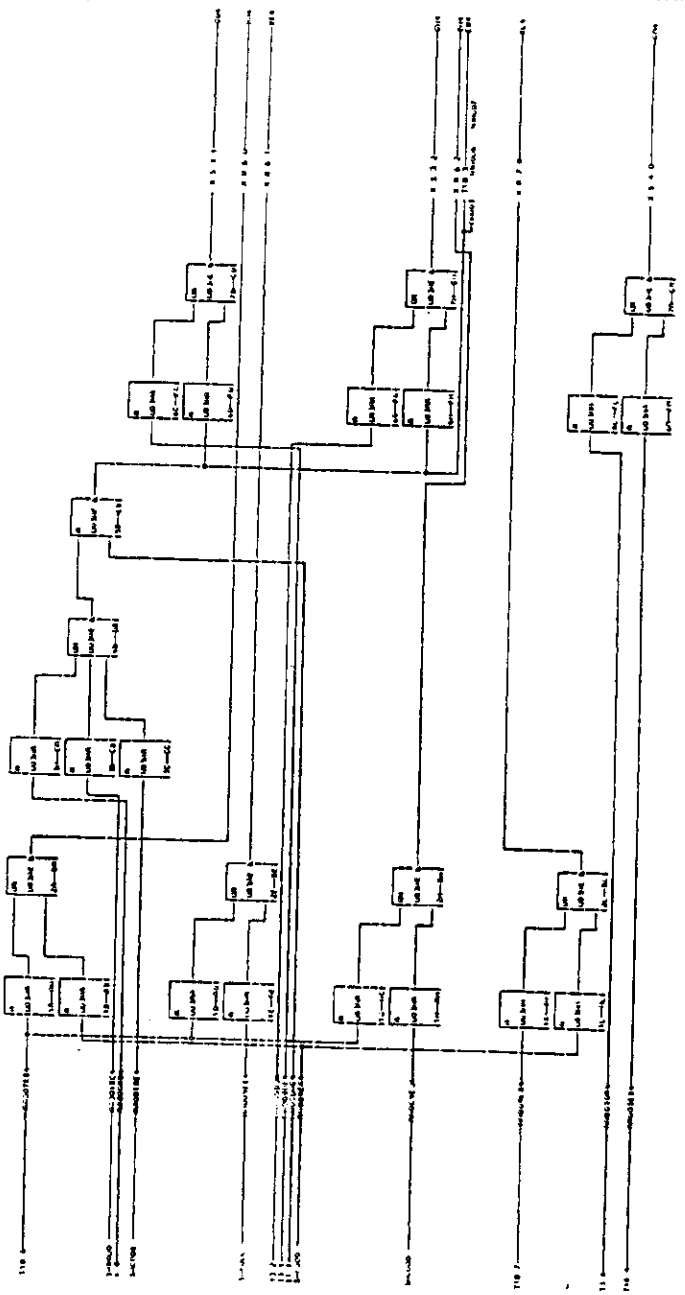
```
T6/0 = K/0
T6/1 = K/1
T6/2 = -K/2
T4/6 = T6/0* T6/1* T6/2
T7/0 = K/0
T7/1 = -K/1
T7/2 = K/2
T4/5 = T7/0* T7/1* T7/2
T8/0 = K/0
T8/1 = -K/1
T8/2 = -K/2
T4/4 = T8/0* T8/1* T8/2
T9/0 = -K/0
T9/1 = K/1
T9/2 = K/2
T4/3 = T9/0* T9/1* T9/2
T10/0 = -K/0
T10/1 = K/1
T10/2 = -K/2
T4/2 = T10/0* T10/1* T10/2
T11/0 = -K/0
T11/1 = -K/1
T11/2 = K/2
T4/1 = T11/0* T11/1* T11/2
T12/0 = -K/0
T12/1 = -K/1
T12/2 = -K/2
T4/0 = T12/0* T12/1* T12/2
T18/0 = T4/0* S1/1
T18/1 = T4/1* S1/1
T18/2 = T4/2* S1/1
T18/3 = T4/3* S1/1
T18/4 = T4/4* S1/1
T18/5 = T4/5* S1/1
T18/6 = T4/6* S1/1
T18/7 = T4/7* S1/1
T15/0/0 = T18/0* T3/0
T15/0/1 = T18/0* T3/1
T15/0/2 = T18/0* T3/2
T15/1/0 = T18/1* T3/0
T15/1/1 = T18/1* T3/1
T15/1/2 = T18/1* T3/2
T15/2/0 = T18/2* T3/0
T15/2/1 = T18/2* T3/1
T15/2/2 = T18/2* T3/2
T15/3/0 = T18/3* T3/0
T15/3/1 = T18/3* T3/1
T15/3/2 = T18/3* T3/2
T15/4/0 = T18/4* T3/0
T15/4/1 = T18/4* T3/1
T15/4/2 = T18/4* T3/2
T15/5/0 = T18/5* T3/0
T15/5/1 = T18/5* T3/1
T15/5/2 = T18/5* T3/2
T15/6/0 = T18/6* T3/0
T15/6/1 = T18/6* T3/1
T15/6/2 = T18/6* T3/2
T15/7/0 = T18/7* T3/0
T15/7/1 = T18/7* T3/1
T15/7/2 = T18/7* T3/2
X/S/0/0 = T15/0/0
X/S/0/1 = T15/0/1
X/S/0/2 = T15/0/2
X/S/1/0 = T15/1/0
X/S/1/1 = T15/1/1
X/S/1/2 = T15/1/2
X/S/2/0 = T15/2/0
X/S/2/1 = T15/2/1
X/S/2/2 = T15/2/2
X/S/3/0 = T15/3/0
X/S/3/1 = T15/3/1
X/S/3/2 = T15/3/2
X/S/4/0 = T15/4/0
X/S/4/1 = T15/4/1
X/S/4/2 = T15/4/2
X/S/5/0 = T15/5/0
X/S/5/1 = T15/5/1
X/S/5/2 = T15/5/2
X/S/6/0 = T15/6/0
X/S/6/1 = T15/6/1
X/S/6/2 = T15/6/2
X/S/7/0 = T15/7/0
X/S/7/1 = T15/7/1
X/S/7/2 = T15/7/2
T17/0/0 = T18/0* -T3/0
T17/0/1 = T18/0* -T3/1
T17/0/2 = T18/0* -T3/2
T17/1/0 = T18/1* -T3/0
T17/1/1 = T18/1* -T3/1
T17/1/2 = T18/1* -T3/2
T17/2/0 = T18/2* -T3/0
T17/2/1 = T18/2* -T3/1
T17/2/2 = T18/2* -T3/2
T17/3/0 = T18/3* -T3/0
T17/3/1 = T18/3* -T3/1
T17/3/2 = T18/3* -T3/2
T17/4/0 = T18/4* -T3/0
T17/4/1 = T18/4* -T3/1
T17/4/2 = T18/4* -T3/2
T17/5/0 = T18/5* -T3/0
T17/5/1 = T18/5* -T3/1
T17/5/2 = T18/5* -T3/2
T17/6/0 = T18/6* -T3/0
T17/6/1 = T18/6* -T3/1
T17/6/2 = T18/6* -T3/2
T17/7/0 = T18/7* -T3/0
T17/7/1 = T18/7* -T3/1
T17/7/2 = T18/7* -T3/2
X/R/0/0 = T17/0/0
X/R/0/1 = T17/0/1
X/R/0/2 = T17/0/2
X/R/1/0 = T17/1/0
X/R/1/1 = T17/1/1
X/R/1/2 = T17/1/2
X/R/2/0 = T17/2/0
X/R/2/1 = T17/2/1
X/R/2/2 = T17/2/2
X/R/3/0 = T17/3/0
X/R/3/1 = T17/3/1
X/R/3/2 = T17/3/2
X/R/4/0 = T17/4/0
X/R/4/1 = T17/4/1
X/R/4/2 = T17/4/2
X/R/5/0 = T17/5/0
X/R/5/1 = T17/5/1
X/R/5/2 = T17/5/2
X/R/6/0 = T17/6/0
X/R/6/1 = T17/6/1
X/R/6/2 = T17/6/2
X/R/7/0 = T17/7/0
X/R/7/1 = T17/7/1
X/R/7/2 = T17/7/2
```

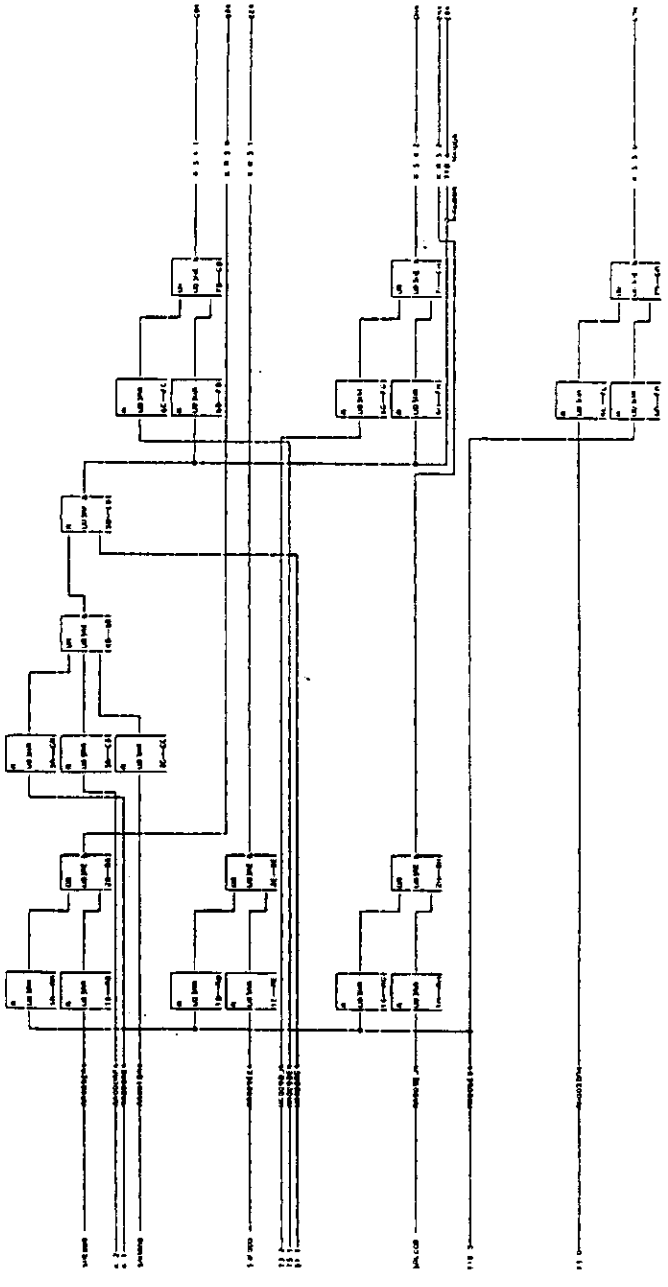
APPENDIX II
LOGIC SHEETS AUTOMATICALLY DIAGRAMMED BY THE IBM LOGIC AUTOMATION SYSTEM

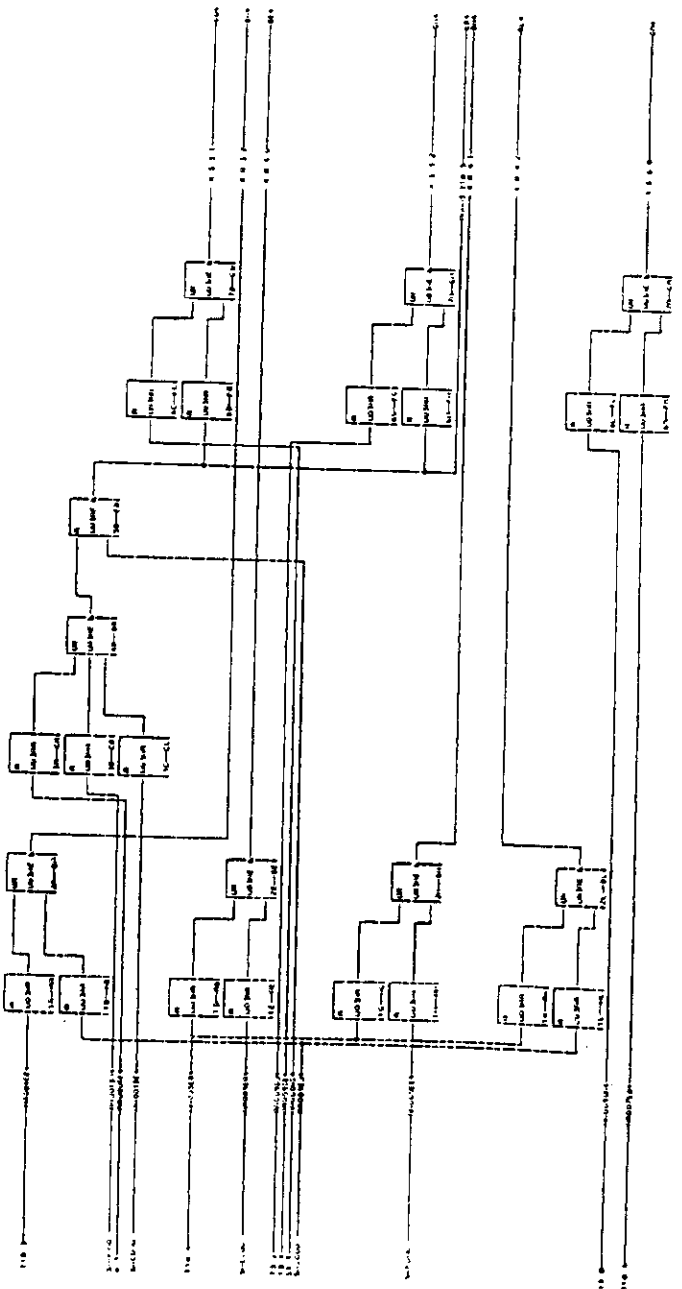


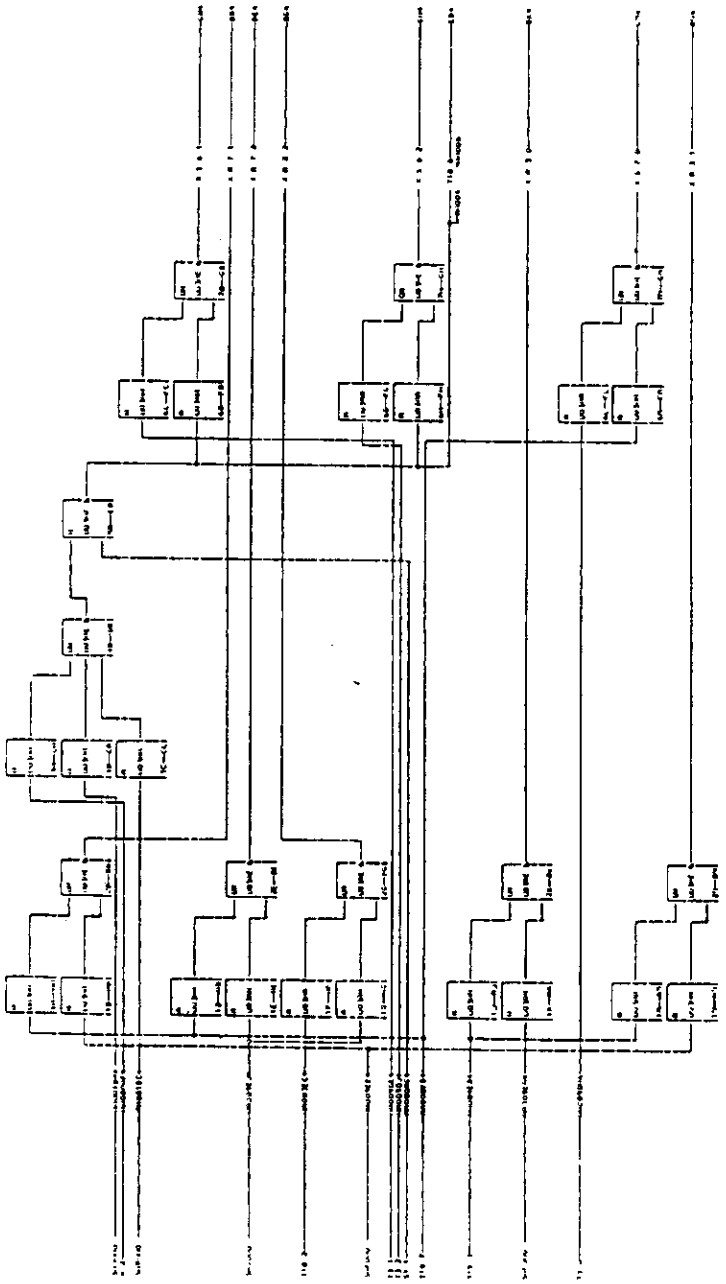


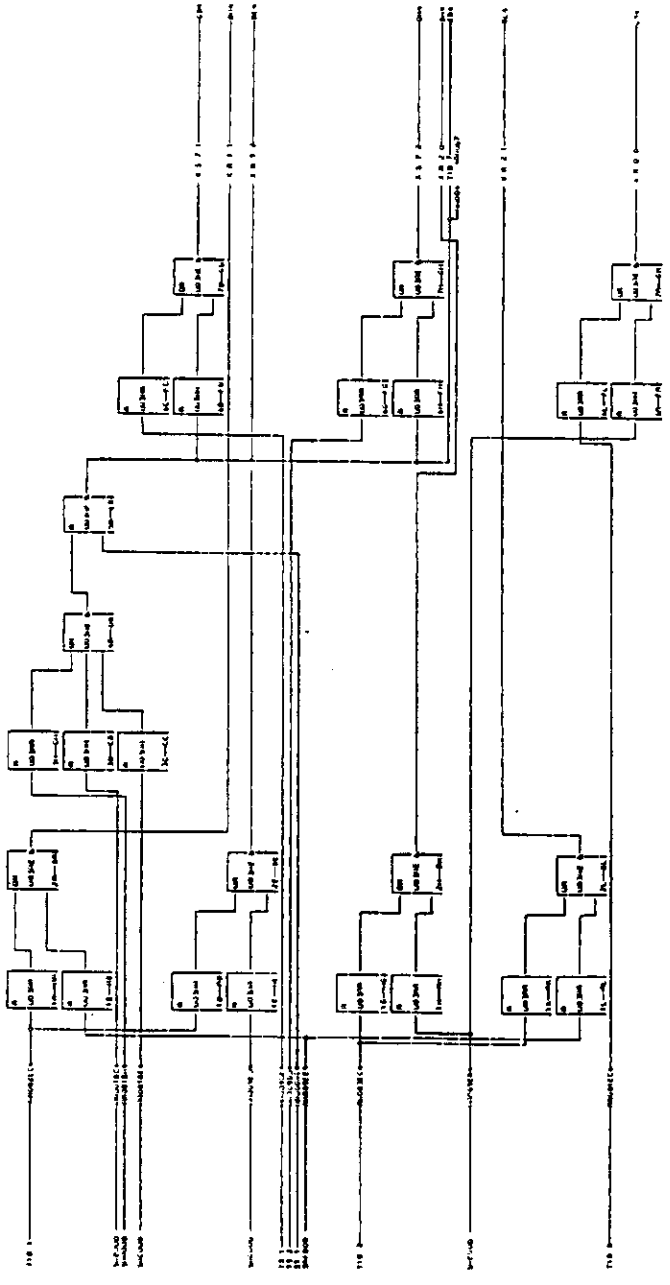


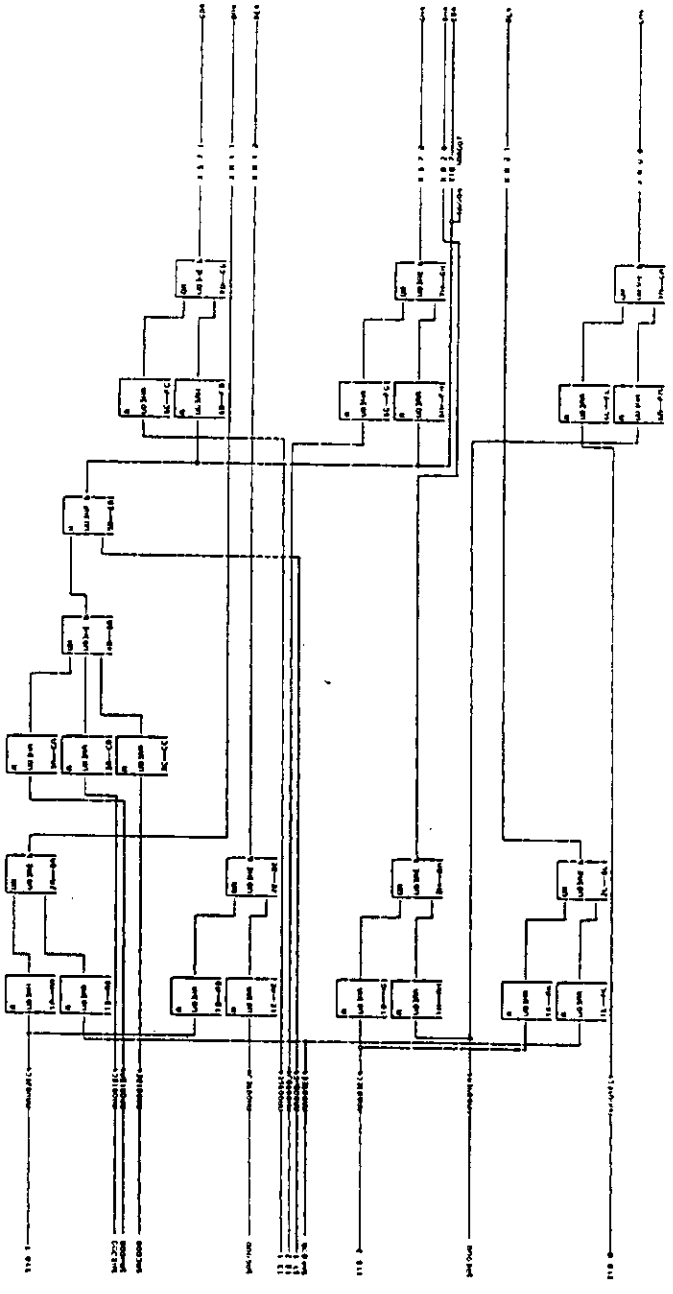












ACKNOWLEDGMENT

The authors wish to th[ank] [C]. D. Coleman, now at IBM Scientific Center, Los Angeles, Calif., for initiating this project.

REFERENCES

[1] P. W. Case, H. H. Graff, L. E. Griffith, A. R. LeClercq, W. B. Murley, and T. M. Spence, "Solid logic design automation for IBM system/360," *IBM J. Res. Develop.*, vol. 8, pp. 127–140, April 1964.
[2] Y. Chu, "An ALGOL-like computer design language," *Commun. ACM*, vol. 8, pp. 607–615, October 1965.
[3] A. D. Falkoff and K. E. Iverson, "APL/360: User's manual," IBM Watson Research Center, Yorktown Heights, N. Y., 1968.
[4] A. D. Falkoff, K. E. Iverson, and E. H. Sussenguth, "Formal description of system/360," *IBM Sys. J.*, vol. 3, pp. 198–262, 1964.
[5] T. D. Friedm[an] [and] C. Yang, "Quality [assurance] from an automatic le[ogic] [gen]erator," IBM Res. Rept. RC 2068, April 15, 1968. (Copies [a]vailable on reque[st].)
[6] D. F. Gorman and J. P. Anderson, "A logic design translator," *1962 Fall Joint Computer Conf., AFIPS Proc.*, vol. 22. Washington, D. C.: Spartan Books, 1962, pp. 251–261.
[7] K. E. Iverson, *A Programming Language*. New York: Wiley, 1962.
[8] G. Metze and S. Seshu, "Computer compiler, part I—Preliminary report," Coordinated Science Lab., University of Illinois, Urbana, Dept. R-364, August 1965.
[9] J. P. Roth, "Systematic design of automata," *1965 Fall Joint Computer Conf., AFIPS Proc.*, vol. 27, pt. 1. Washington, D.C.: Spartan Books, 1965, pp. 1093–1100.
[10] H. P. Schlaeppi, "A formal language for describing machine logic, timing, and sequencing (LOTIS)," *IEEE Trans. Electronic Computers*, vol. EC-13, pp. 439–448, August 1964.
[11] H. Schorr, "Computer-aided digital system design and analysis using a register transfer language," *IEEE Trans. Electronic Computers*, vol. EC-13, pp. 730–737, December 1964.

EXHIBIT 5
(Part 3 of 4)

AS-6

Experiments in Logic Synthesis

John A. Darringer
William H. Joyner
Leonard Berman
Louise Trevillyan

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

VINCENT N. TRANS
EXAMINER
ART UNIT 25

Abstract

The rise in circuit density and processor complexity is driving up the cost of engineering changes and making it increasingly important to insure that initial chip implementations are correct. No longer is the logic designer concerned solely with minimizing circuit count. Instead, he is faced with a growing number of design requirements and technology restrictions. This paper describes experiments with an automated system that allows a designer to start with a naive implementation produced automatically from a functional specification, evaluate it with respect to these many factors, and incrementally improve this implementation by applying local transformations until it is acceptable for manufacture. The use of simple local transformations in this system insures correct implementations, isolates technology-specific data, and allows the total process to be applied to larger, VLSI designs.

Background

The goal of logic synthesis is to accept functional specifications for a hardware unit and to generate automatically a detailed, technology-specific implementation comparable in quality to that of an experienced engineer. The nature of this problem depends on the level of the functional description, the set of implementation primitives, and the criteria of acceptability. Early work centered on developing algorithms for translating a boolean function into a minimum two-level network of boolean primitives. Extensions were developed for handling limited circuit fan-in and alternative cost functions [1, 2]. But because these algorithms search for minimal implementations they are necessarily exponential and require too much space and time to be used on most actual designs.

Later efforts attempted to raise the level of specification. The DDL work at Wisconsin [2, 3, 4], APDL at Carnegie-Mellon University [5], and ALERT at IBM all began with behavioral specifications and produced technology-independent implementations at the level of boolean equations. The results were usually more expensive than manual implementations and did not take advantage of the target technology. For example, the ALERT system was validated on an existing design, the IBM 1800, and the implementation produced required 160% more circuits than the manual design [6].

Several tools have been developed at Carnegie-Mellon University to support the early part of the design cycle [7, 8, 9, 10]. In one experiment [11] the CMU-DA system was used to implement the data path portion of a PDP-8/E. They began with a functional description of the machine and produced an implementation in two technologies of the registers, register operators, and their inter-connections, but not the control logic to sequence the register transfers. When the target technology was TTL series modules their design required 30% more modules than the DEC implementation. When it was CMOS standard cells they required 150% more area than an existing Intersil chip.

We are concentrating on the control portion of a machine, since its design is more error prone than data path design. Thus we assume that all memory elements of the final implementation are identified in the specification. Also we are focusing on "random logic" implementations, instead of generating microcode for a control processor or using a programmable logic array. Our implementations are interconnections of primitives selected from a specified set, and connected to satisfy given performance requirements and technology restrictions.

The Approach

In a previous paper [12] we described what we believe to be a new approach to this form of synthesis. We are not proposing a completely automatic replacement for the manual design process. Instead, we envision an interactive system in which the user operates on a logic design at three levels of abstraction. He begins with a initial implementation generated in a straightforward manner from the specification. He can simplify the implementation at this level, and when satisfied can move to the next level. He does this by applying transformations, either locally or globally, to achieve the simplification or refinement. By being able to operate on the implementation at several levels, the user can often make a small change at one level that will cause a larger simplification at a lower level. By limiting the user to directing function-preserving transformations, we can insure that in all cases the implementation produced will be functionally equivalent to the specified behavior.

Logic synthesis is a problem of finding a feasible (not an optimal) implementation: a network of primitive boxes that satisfies a large number of constraints. In addition to gate and I/O pin limitations, there are timing constraints, a restricted library of primitives, driver requirements, clock distribution rules, fan-in and fan-out constraints, and rules for testability. Since we eventually hope to apply our techniques to VLSI chips, we are attempting to limit our transformations to local changes that do not require time or space exponential in the size of the chip.

CH1511-5/80/0000-0234 $00.75©1980 IEEE

RCL000142

A Prototype System for Logic Synthesis

The organization of the logic synthesis system is shown in Figure 1. Its inputs are the register transfer specification, the interface constraints, and the technology file which characterizes the target technology. The output is a detailed implementation in terms of the primitives of the target technology, which is submitted to placement and wiring programs for physical design. After placement and wiring, some excessively long paths may become apparent. In this case the synthesis process is repeated with a revised specification or modified constraints until an acceptable implementation is achieved.

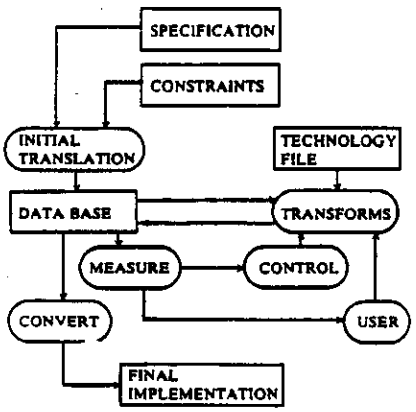


Figure 1. The Logic Synthesis System

The Logic Synthesis Data Base

An important requirement of our approach is that the data base be capable of representing the implementation at different levels of abstraction. Our system to support logic synthesis makes use of a graph-like internal data structure for storing the implementation as it progresses from the higher-level description to its final form, and all transformations operate on this graph. There is a single organizational component: the 'box'. A box has input and output terminals which are connected by wires to other boxes. Each box also has a type, which may be primitive or may reference a definition in terms of other boxes. Thus a hierarchy of boxes can be used, and an instance of a high-level box such as a parity box can be treated as a single box or expanded into its next level implementation when that is desirable.

The Logic Synthesis data base is implemented using a system originally developed for use in an experimental compiler project within IBM Research [13]. It is made up of two groups of tables. The first group describes the technology being used. For each box type in the technology, the tables contain information such as name, function, and number and names of input and output pins. This data is created in batch mode and is read during initialization of the interactive system.

The second group of tables contains the representation of the logic created by the interactive system. This group consists of a box table, a signal table, and a set of auxiliary tables which describe the relationship between the boxes and the signals. There is some intentional redundancy in the data; each box has a complete list of input and output signals, and each signal has a source and a list of sinks. Every box table entry contains type information which provides a link to the technology group. This allows programs to get technology information about a specific box.

Transformations communicate with the database via a layer of functions which perform all data addition, retrieval, and deletion. They also provide the transformations with the ability to traverse a chip by following signal paths, or by visiting each box. The functions provide a conceptual view of the database which remains stable even when the database is altered. The table structure representing this view can be significantly changed with a minimal impact on the processing programs.

The Experiments

The first three experiments with the Logic Synthesis System were attempts to produce implementations for chips from existing processors that had been specified functionally and implemented by engineers. The existence of the engineers' implementations permitted comparison of designs and a study of the differences between manual designs and those produced automatically. The fourth experiment was a use of the same system to transform an existing chip implementation into a new technology. Each of the experiments was carried out automatically, although the particular application sequence of transformations was the result of much experimentation.

Experiment 1

For our first experiment we selected a chip that was characterized as "straightforward". The specification described 7 registers totaling 24 bits, 2 parity operators, and the conditions for the data transfers. The target technology was a TTL masterslice that provided 96 I/O pins and 704 NAND gates on each chip. In addition to the NAND gates, there are a number of macros such as receivers, senders, and flip-flops that are implemented with these NAND gates. Restrictions on the use of the primitives available such as fan-in and fan-out requirements, timing constraints, clocking and powering rules, etc., were described in the technology file or in some cases built into the transformations.

Our objective was to find a set of transformations and a sequence for applying them such that the original functional specification could be transformed by a sequence of small steps into an acceptable implementation. After examining many alternative scenarios, we arrived at the one shown in Figure 2. A similar scenario was used in all the synthesis experiments, with some differences that will be discussed later. We also spent considerable time talking with the designer of this chip to understand the motivation for the many design decisions.

Our strategy is to generate an naive implementation from the functional specification and then to perform local simplifications at three levels of abstraction: the AND/OR level, the NAND level, and the hardware level. The initial implementation is produced by merely replacing specification language constructs with their equivalent AND/OR implementations. Methods for this translation have been described in [3, 5]. Transformations are used at each level to simplify the implementation according to appropriate measures and to move the implementation from one level to the next. The transformations are local in that they

replace a small subgraph of the network (usually five or fewer boxes) by another subgraph which is functionally equivalent but simpler according to some measure.

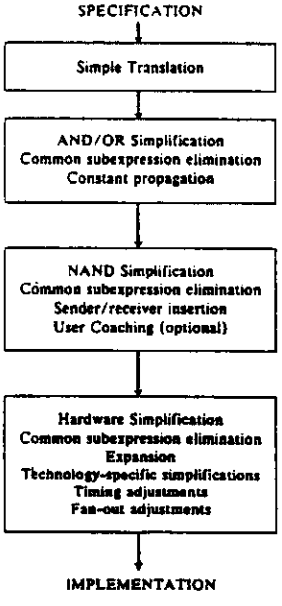


Figure 2. The Scenario for Synthesis

At every level the implementation is a network of boxes connected by signals. At the first level the boxes are of types such as AND, OR, NOT, PARITY, or REGISTER. Simple local transformations are applied to reduce the number of boxes. The particular transformations used are listed below:

NOT(NOT(a)) → a
AND(a,NOT(a)) → 0
OR(a,NOT(a)) → 1
OR(a,AND(NOT(a),b)) → OR(a,b)
XOR(PARITY(a1,...,an),b) → PARITY(a1,...,an,b)
AND(a,1) → a
OR(a,1) → 1

The last two simplifications are examples of a more general constant propagation that is performed. This action may leave fragments of logic disconnected. We clean up the dead logic in a manner similar to the way compilers perform dead code elimination. Another technique from optimizing compilers, common subexpression elimination, is also applied here and at other points in the synthesis process to further reduce the size of the implementation.

Next the AND, OR, NOT, and most other operators of the initial description are replaced by their NAND implementations. NANDs were selected for this level because they are the primitives available in this particular technology. This transition also is accomplished by local transformations, and may introduce unnecessary double NANDS, which will be eliminated later. Also at this point, the chip interface information is used to place generic (i.e., not technology-specific) senders and receivers on the chip inputs and primary outputs, and to insert NANDs where necessary to insure the correct signal polarities. Higher-level operators such as EQ and PARITY are not replaced at this point but will be expanded at the hardware level.

Simplifying transformations are now applied to each signal in the network at the NAND level. These NAND transformations attempt to reduce the number of boxes of the implementation without increasing the number of connections. To accomplish this, the transformations must check the fan-out of the various signals involved, since this will affect the number of boxes and signals actually removed. The transformations are applied repeatedly throughout the network until no more apply. Figure 3 illustrates the NAND transformations used in this experiment.

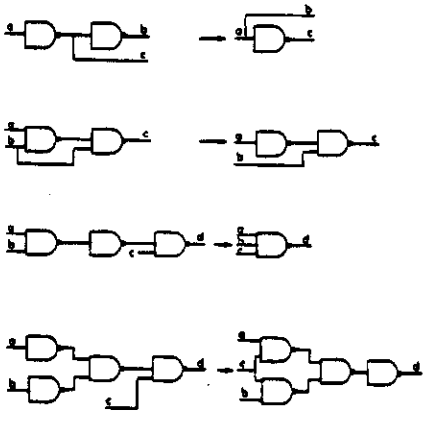


Figure 3. The NAND Transformations

In examining the implementation after the NAND transformations were applied, it was noticed that further improvements could be made. In particular a reduction in fan-out of a signal by repowering its source would allow a transformation to apply and eventually reduce the size of the implementation. The system allows repowering and some other transformations to be applied to particular signals, rather than across the whole implementation, as a form of user "coaching". In this instance coaching saved only four boxes, but that resulted in an implementation slightly better than the manual one.

In the transition to the hardware level the NAND gates and generic registers are replaced by technology-specific primitives. Single primitives or macros are selected to match the fan-in of the actual primitives with that of the "idealized" boxes. Also the number of control and data lines of the idealized registers might exceed those normally available, and necessitate the generation of additional logic. At this point the implementation is in terms of primitives used by the engineers in their implementations, but because transformations have been made locally, there may be some violations of timing, fan-out, and other technology restrictions.

The simplifying transformations at the hardware level are of two sorts. Some are simplifications similar to those at the previous levels, such as eliminating the equivalent of double NOTs, which may occur as a result of expanding higher level boxes, such as EQ and PARITY. Others attempt to take advantage of the particular technology. For example, flip-flops in this technology provide an output and its complement allowing some inverters to be removed at this level. Also, because of combination flip-flop-receiver books available, some receivers may be eliminated. Wired or dotted ANDs can be introduced to reduce cell count where possible. Other technology-specific transformations applied at this level distribute clock signals to flip-flops according to the technology rules, eliminate long and short paths between flip-flops (assuming a unit gate delay and technology-specific guidelines), and adjust fan-out by repowering signals.

Results -- the first experiment resulted in a synthesized implementation that was remarkably similar to the manual one. In fact, it required four fewer cells, five fewer connections, and four shorter paths than the engineer's implementation. The similarity, however, was not such a surprise since we had used this example in the design of our system, and since we had worked so closely with the chip's designer.

Experiment 2

For our second experiment we wanted to try the same sequence of transformations on a more complex chip. The chip specification we selected contained 13 register bits, a 3 bit counter, a 5 bit counter, 2 parity operators, and more complex conditions controlling the data transfers. The target technology was the same as in the first experiment. Also this time there was virtually no contact with the engineer who designed the chip.

While we tried to use the same scenario, we did make two changes. There was no coaching in this experiment and counters were handled differently from the EQ and PARITY in the first experiment. We found that it is better to expand the counters at the AND/OR level, than at the hardware level. This exposes the expanded counter to all subsequent simplifications and allows one definition to be used for different technologies. The expansion transformation therefore has been extended to permit expansion of a nonprimitive box at any level.

Results -- the synthesis of the second chip resulted in a implementation with 15% more cells and 20% more connections than the manual implementation. We are currently analyzing these results to understand why our implementation is more complex.

Experiment 3

The third experiment was an attempt to synthesize another complex chip in a different technology. This third chip specification described 28 register bits, 3 parity operators, 4 decoders, 7 comparitors, and even more complex control logic. The target technology was an ECL masterslice. In addition to a new set of technology rules and restrictions, this means that the basic primitive was a NOR and that each primitive had "dual-rail outputs", that is it provides both polarities of its output. The synthesis scenario was adapted to this technology and changed slightly, but the three levels of implementation were maintained. The decoders and comparitors were expanded at the AND/OR level and the AND/OR transformations remained unchanged. Common subexpression elimination was applied more often at this level and throughout the scenario.

The NAND level became the NOR level because of the new technology. This required a new transformation to translate the AND/OR primitives into NORs, and a set of NOR simplifications. These were originally just the NAND transformations with the NANDS converted to NORs, but we later realized that with dual rail outputs, an apparent box saving at the NOR level might not be a saving at the hardware level, and that the transformation might increase fan-in or number of connections. Thus different fan-out restrictions were used in the NOR transforms. At the hardware level new definitions for PARITY and EQ, were written. The technology-specific transformations had to be rewritten for the new technology, and some new ones were added, such as the one to eliminate inverters.

Results -- this experiment resulted in a implementation with 10% more boxes than the manual one. We are trying to account for this additional logic and determine if it could be eliminated through local transformations.

Experiment 4

In addition to the 3 synthesis experiments we extended the system to explore transforming a chip implementation from one technology to another (TTL masterslice to masterslice ECL as an example). This required two new transformations, one that transformed primitives at the hardware level back to the NAND level, and a second that transformed the NAND implementation into a NOR one, while preserving the chip input/output behavior. This approach is better than the straightforward replacement of old technology primitives by new ones, since it exposes the remapped implementation to the simplifications at the NOR level and at the hardware level.

Results -- Unfortunately, this chip conversion was not performed manually so we could not make an objective comparison. We did check that the input/output behavior was preserved and showed the implementation to an experienced engineer who found no serious problems.

Comparing Implementations

One of the problems that confronts us is the difficulty of evaluating the result of the synthesis process. In our work to date, this evaluation has meant a comparison between our generated implementation and a manually produced implementation. There are two aspects to the comparisons that we must perform. One is the problem of determining functional equivalence between the two implementations. The other is to furnish a response to the ill-posed question: "How do these implementations differ?"

Functional equivalence in its full generality is the problem of boolean equivalence and known to be co-NP complete. This implies that at our present level of understanding, it is not possible to devise a program which will efficiently, in all cases, decide equivalence between two implementations. We cannot solve this problem; but we are exploring heuristics which may offer significant speed-up on a large class of implementations. A report on this work is in preparation.

Even when two implementations are functionally equivalent, we are still interested in their structural similarity. This form of comparison permits us to evaluate stylistic difference between our implementation and that produced by an engineer. This is necessary for discovering new heuristics. For this form of comparison we are considering formalizing the notion of "distance" between two implementations, following an analogy to the spelling correction problem.

RCL000145

The Future

During the coming months we plan to continue analyzing the results of our experiments to determine what improvements should be made to our system. We will also look at more ambitious chips -- chips that have required minimization or that have caused long path problems, when implemented manually. We hope to arrive at a set of measures and transformations that will provide acceptable implementations for a large class of examples. In addition, we will explore:

- multi-chip synthesis -- starting with a functional specification that requires several chips, developing additional measures and transformations that will trade resources across chip boundaries.
- engineering changes -- examining how such a synthesis system could respond to EC's where minimum, local changes are highly desirable.
- transformation specification -- looking at how transformations could be described at a high level and compiled for efficient application.
- transformation correctness -- considering what properties of transformations such as function-preserving should be proved and demonstrating how such proofs can be accomplished.

Summary

We are in the process of exploring what we believe is a new approach to the old problem of logic synthesis and are encouraged by our initial experiments. We have built a prototype synthesis system and used it to synthesize 3 masterslice chips, requiring 0%, 15%, and 10% more logic than their respective manual designs. We have also used our system to remap an implemented chip into a new technology, while preserving its input/output behavior. We plan to perform further experiments, to study the remaining differences between the automatic and manual implementations, and to improve the competence of our prototype system. Our hope is that computationally manageable techniques based on local transformations can be applied to improve naive implementations to acceptable ones. This could greatly shorten processor development and validation times.

Acknowledgement

We would like to thank William van Loo and James Zeigler for many helpful discussions on masterslice chip design. Also John Gerbi, Thomas Wanuga, and Alan Stern have made valuable contributions to the design and implementation of the prototype synthesis system.

References

[1] Breuer M A (Editor), *Design Automation of Digital Systems*, Prentice-Hall, 1972.

[2] Dietmeyer D L, *Logic Design of Digital Systems*, Allyn and Bacon, 1971,1978.

[3] Duley J R, "DDL -- A Digital Design Language", Ph.D. Thesis, University of Wisconsin 1968.

[4] Duley J R and D L Dietmeyer, "Translation of a DDL Digital System Specification to Boolean Equations", *IEEE TC* Vol C-18, April 1969.

[5] Darringer J A, "The Description, Simulation, and Automatic Implementation of Digital Computer Processors", Ph.D. Thesis Carnegie-Mellon University 1969.

[6] Friedman T D and S C Yang, "Quality of Design From an Automatic Logic Logic Generator(ALERT)," Proc. 1970 Design Automation Conference.

[7] Barbacci M, "Automated Exploration of the Design Space for Register Transfer Systems", PhD Thesis, CS Dept., Carnegie-Mellon University, 1973.

[8] Thomas D E, "The Design and Analysis of an Automated Design Style Selector", PhD Thesis, EE Dept., Carnegie-Mellon University, 1977.

[9] Snow E A, "Automation of Module Set Independent Register-Transfer Level Design", PhD Thesis, EE Dept., Carnegie-Mellon University, 1978.

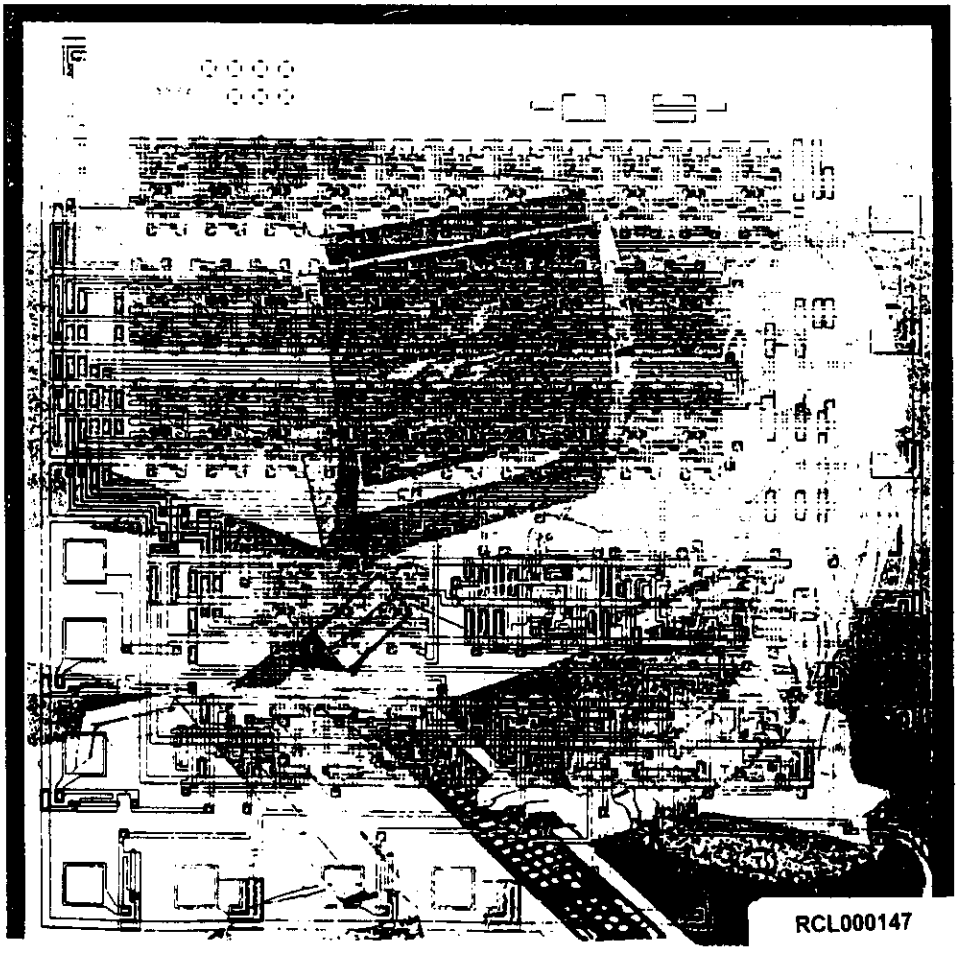
[10] Hafer L J and A C Parker, "Register-Transfer Level Digital Design Automation: The Allocation Process", Proc. 15th Design Automation Conf., 1978.

[11] Parker, A., et.al., "The CMU Design Automation System -- An Example of Automated Data Path Design", Proc. 16th Design Automation Conf., 1979.

[12] Darringer J A, and W H Joyner "A New Approach to Logic Synthesis", 17th Design Automation Conf. June 1980.

[13] Allen F, et.al "The Experimental Compiler System", IBM Journal of Research and Development, Nov. 1980.

Conference Publication Number 232



European Conference on

Electronic Design Automation (EDA84)

26-30 March 1984

Organised by
The Electronics Division of the Institution of Electrical Engineers

in association with the
British Computer Society (BCS)
Convention of National Societies of Electrical Engineers of Western Europe (EUREL)
Institute of Physics (IoP)
Institution of Electronic and Radio Engineers (IERE)
Institution of Mechanical Engineers (IMechE)
International Federation for Information Processing (IFIP)

Venue
University of Warwick

RCL000148

Contents

The Institution of Electrical Engineers is not, as a body, responsible for the opinions expressed by individual authors or speakers

Page No.

THE ROLE OF DESIGN AUTOMATION

Invited Address

1 Gate Arrays - Computer aids automation and the UK5000
John R Grierson,
British Telecom UK

5 The management of electronic design automation
T S McLeod
Plessey Controls Ltd. UK

SIMULATION AND CIRCUIT ANALYSIS

9 The design of an hierarchical circuit-level simulator
M Zwolinski and Professor K G Nichols
University of Southampton, UK

13 A circuit analysis program using an attached array processor
T Kage, Miss Y Oishi and M Ishii
Fujitsu Labs, Japan

18 Efficient interpretive code for sparse matrices arising in CAD for large scale systems
Dr D G Agnew
Northern Telecom Electronics, Canada

23 An eight-value modelling technique for logic simulation of transmission Gates and Buses
C B Almeida and Professor M J A Lança
CEAUTL - Instituto Superior Tecnico, Portugal

VLSI DESIGN SYSTEMS

28 Systematic design methodology and performance verification for digital macro systems
J S Saini and E J Zaluska
University of Southampton, UK

33 An approach to VLSI logic design
L F Saunders
IBM, USA

35 Custom CMOS design using the Astra CAD system
Dr P A Ivey and Dr M C Revett
British Telecom Research Laboratories, UK

40 On the integration of a CAD system for IC design
Dr E M da Costa
Centre for Research and Development - TELEBRAS, Brazil

Page No.	
	FROM TESTING TO MANUFACTURE
46	HITEST test generation system - user display interfaces *Dr B D V Smith* *Cirrus Computers Ltd, UK*
51	Integrating computer aided design and test *P Jennings* *Marconi Instruments Ltd, UK*
54	Auto PL/T *T J Kristek, R E Kizis and G C Wickham* *IBM, USA*
59	Computer aided equipment for prototyping thick film hybrid circuits *J F Herington, A G Saunders and R J Howells-Harris* *British Telecom Research Laboratories, UK*
	SYSTEMS DESIGN AND DATA MANAGEMENT
63	Integration tool for computer-aided design systems (ICADS) *P J Horth and K J Baker* *British Telecom, UK*
68	Data management in the DAX design automation system *D C Harwood* *ICL, UK*
	TEST AND TESTABILITY OF SYSTEMS AND CIRCUITS
72	Petri-Net test generation on systems *T Alukaidey and Professor G Musgrave* *Brunel University, UK*
79	A behavioural test method for microprocessors and complex circuits *C Bellon and R Velazco* *Laboratory IMAG, France*
83	Fault effects in MOS circuits and their implications for digital circuit testing *D R J Wilkins and S Shaw* *British Telecom Research Labs, UK* *N Burgess and Dr R I Damper* *University of Southampton, UK*
	LAYOUT FOR VLSI
92	Describing integrated circuit layouts in pascal: techniques and conclusions *S S Olay and Professor D J Kinniment* *University of Newcastle upon Tyne, UK*
97	SYLAM: an adaptive symbolic layout system *Dr C Landrault, S Pravossoudovitch and A Rellinger* *Université des Sciences et Techniques du Languedoc, France*
102	Logical and topological design in MOS technology *Professor G Saucier and Mrs G Thuau* *Laboratory IMAG, France*

Page No.	
	DESIGN VERIFICATION OF SYSTEMS
107	A multilevel, mixed state simulator for hierarchical design verification *S Hodgson* *ICL, UK*
111	Computer-aided design of micro-systems *A D Ivannikov and Professor P P Sipchuk* *Moscow Institute of Electronic Machine Building, USSR*
115	Timing evaluation of logic design *A Kay and C Oldland* *ICL, UK*
	DESIGN AUTOMATION WORKSTATIONS AND HARDWARE
120	A front end graphic interface to the FIRST silicon compiler *J H Nash and S G Smith* *University of Edinburgh, UK*
125	An engineers design workstation for the autolayout of ULA gate arrays *F R Ramsay* *Ferranti Electronics Ltd, UK*
129	Computer aided engineering and engineering workstations at British Aerospace plc, DG Stevenage Division *R D Roe* *British Aerospace plc, UK*
	GATE ARRAY SYSTEMS
134	Texas Instruments transportable design utility - TDU C K Thomas Texas Instruments Ltd, UK
138	A microelectronic gate array designed to give efficient automated circuit implementations *A McDonald* *University of Bath, UK* *P Jennings* *Imperial College, UK*
143	A functional testable design of programmable logic arrays *Professor G Musgrave* *Brunel University, UK* *Chongxun Zheng* *Xi'an Jiaotong University, People's Republic of China*
	DESIGN VERIFICATION OF VLSI
147	CADOC: a functional specification and simulation tool for VLSI *P Amblard, M Crastes de Paulet, J Rarivomanana and Professor G Saucier* *Laboratory IMAG, France*
152	Architecture of a conversational symbolic simulator for the digital circuit design *Dr E Chouraqui and L Bourrelly* *LISH-CNRS, France* *Professor N Giambiasi* *LAIM-Université d'Aix-Marseille, France*
156	Timing verification of large digital circuits *Mrs I M C Teixeira and Professor M J A Lança* *CEAUTL - Instituto Superior Tecnico, Portugal*

RCL000151

Contents

Page No.	
	TESTING AND TESTABILITY OF VLSI
161	One-chip microcomputer design based on isochronity and selftesting *L Spaanenburg* *Twente University, The Netherlands* *P B Duin* *Philips-Elcoma, The Netherlands* *R Woudsma* *Philips Central Research Labs, The Netherlands* *A Van der Poel* *ICN, The Netherlands*
166	Intelligent assistance for test program generation *Dr C Bellon and Professor G Saucier* *Laboratory IMAG, France*
171	Computation of the critical area in semiconductor yield theory *Dr A V Ferris-Prabhu* *IBM General Technology Division, USA*
	PLACEMENT FOR VLSI
174	An expert chip planning tool *Professor P Antognetti, A De Gloria, L Repetto and F Allena* *Istituto di Elettrotecnia, Italy*
179	Parallelism in the placement problem in structured logic *Professor G Saucier and P Chaisemartin* *Laboratory IMAG, France*
184	Classification and comparison of different placement strategies *A Bellon, Professor G Saucier, J F Paillotin, J Tsitsinis and A. Janati* *Laboratory IMAG, France*
	SPECIFICATION DESIGN LANGUAGES
189	A hierarchical hardware description language M Miyata, I Yamazaki and S Nishio Toshiba Corporation, Japan
194	A concept for computer hardware description on the register transfer level *K Kuchcinski* *Institute of Computer Science Politechnika Gdanska, Poland*
199	A VLSI design language incorporating self-timed concurrent processes *D J Allerton, D A Batt and A J Currie* *University of Southampton, UK*
204	HACK: a hardware compilation kit *S J Baker, S F Bryant and R A Cook* *Philips Research Laboratories, UK*

A FRONT END GRAPHIC INTERFACE TO THE FIRST SILICON COMPILER

J H Nash* and S G Smith

University of Edinburgh, UK

ABSTRACT

A graphic front end is described that performs as an input interface and knowledge base to the FIRST silicon compiler. The method of implementation is described along with a discussion of the many advantages graphic input has over language input to such a Computer Aided Design (CAD) system. Future work to provide a "silicon compiler workbench" is also discussed.

INTRODUCTION

Input to silicon compilers is usually via some high level language. This is true of the FIRST [1] silicon compiler developed at the University of Edinburgh for the production of Integrated Circuits for signal processing.

Language as an interface can be full of problems due to the specification of a syntactically correct set of input data but an incorrect circuit description. This inaccuracy stems from the translation of a hand drawn circuit into the high level language that drives the compiler. A graphic interface that can eliminate these errors at an early stage is therefore not just a much more "user friendly" system but also a much more complete system.

The work to be described removes this level of inaccuracy by allowing the user to draw his circuit using an intelligent graphics editor written specifically as a front end to the FIRST compiler. The editor has all the features of a two dimensional drawing system[2] , allowing for complete hierarchical nesting of the symbolic layout that in turn generates the FIRST language. The editor also carries out real time checks on the data as it is input, thus finding design errors early in the design process (making for quicker redesign cycles), rather than waiting for the compiler itself to flag the errors. The knowledge base that the editor accesses to aid the user can also be used to give on-line help information to the novice designer. The data is of course linked to the main database that drives the compiler in generating layout and geometry.

The position of the graphics editor within the FIRST environment is explained as well as its implementation. The graphics front end editor is expanded upon with relation to the drawing operations and the relationship to the knowledge base is explained.

Examples of a design carried out by hand and by the front end are compared to give an idea of the different interface experienced using graphics as opposed to text as an input medium.

The FIRST Environment

Figure 1 shows the FIRST compiler as described elsewhere[1] with the three main processes clearly visible.

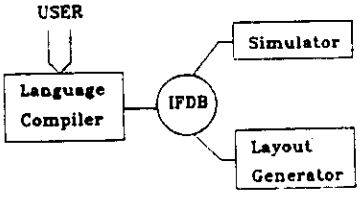
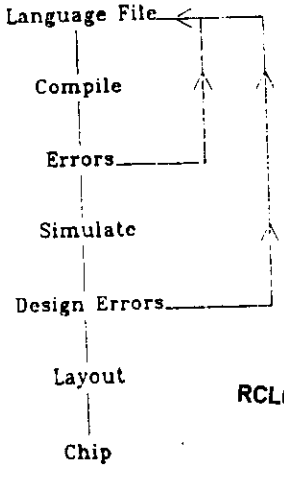


Figure 1. The FIRST Silicon Compiler

The compiler takes a language input and converts this into an Intermediate Form DataBase (IFDB) having carried out syntactic and semantic checks. The IFDB then drives both the simulator and the layout generator that produces the chip. The usual design cycle is is shown in Figure 2.



RCL000153

Figure 2. The Usual Design Cycle

* Formerly with the Integrated Systems Group, Department of Electrical Engineering, University Of Edinburgh.

Once all the syntax errors have been eliminated along with errors produced by incorrect description, the simulator is used to verify the final design. Once the simulations are satisfactory the user can then generate a IC chip with confidence that when fabricated it will perform to the simulated specification.

The graphic interface to FIRST is shown in Figure 3 and the implications to the design cycle shown in Figure 4.

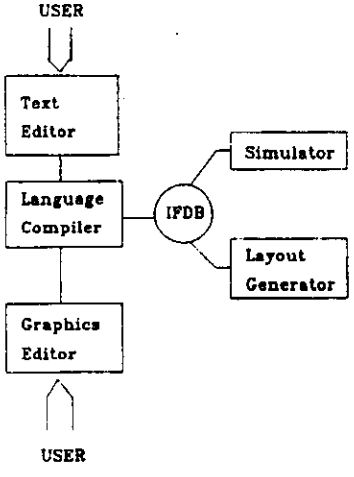


Figure 3. FIRST plus Graphic Interface

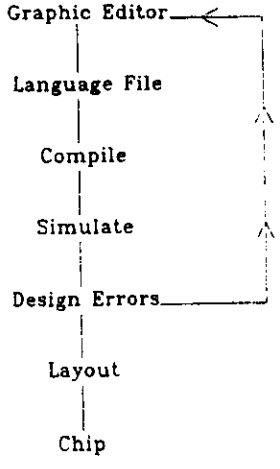


Figure 4. Graphic Design Cycle

As can be seen the editor produces correct language that will never contain errors. The editor picks up errors in real time and will not allow syntactically incorrect designs to proceed. The language that is produced is for reference only as the editor could just as easily produce the IFDB directly. The elimination of the language is thought undesirable as this would remove an avenue of approach that some designers prefer to use, and to provide two independent paths to the IFDB is also undesirable.

Figure 5 shows the full FIRST system that includes the links to the knowledge base and the simulation graphics programs. The links with the user are indicated, this complete system can be described as a "silicon compiler workbench" although the simulation and layout generation are not included in the integrated system as yet and have to be approached as stand alone programs. The aim is to integrate the whole system to give a user full command of all the functions within the "workbench" from one simple graphics front end interface. A start has been made with the interface to the language compiler as described in this paper, the rest will follow.

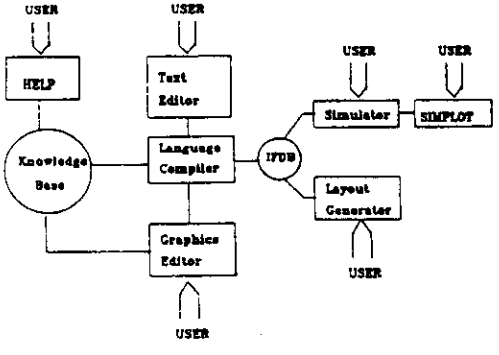


Figure 5. Compiler "Workbench"

THE GRAPHICS EDITOR

The hardware configuration used is a raster colour display with an alphanumeric slave screen attached. An optional tablet is also available. A serial line connects to a VAX 11/750 running UNIX[3] with all the code written in 'C'[4]

The hardware is shown in Figure 6. The Colour terminal is used for the graphics and the alphanumeric slave screen is used for textual interaction and help information. Input is via the keyboard or tablet. The layout of the Graphics screen is shown in Figure 7. The main area is for composing the design, the small menu boxes represent the main options

```
EXIT ----- quit the program
FIRST ---- generate FIRST language
           of current display
HELP ----- enter the HELP program
COMPILE -- generate the IFDB of the
           current display
SIMULATE - Enter the Simulation
           package (unimplemented)
CHIP ----- enter the IC layout
           package (unimplemented)
```

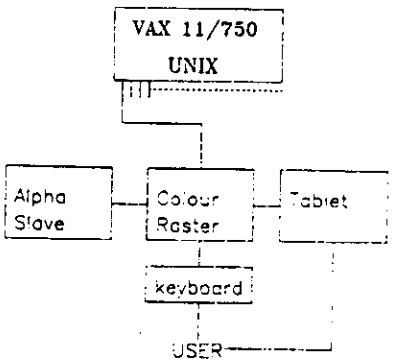


Figure 6. Hardware Configuration

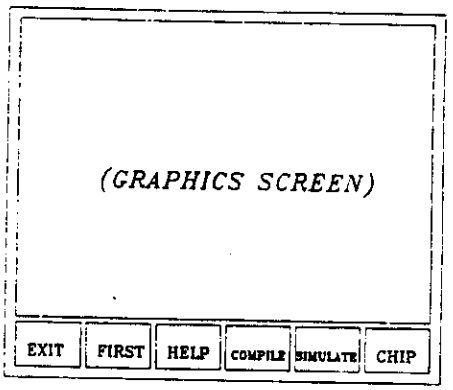


Figure 7. Screen Layout

The unimplemented parts of the above menu refer to parts of the system that still have to be run as stand alone processes. The general graphic operations of the editor are carried out using keyboard or tablet menu hits as appropriate.

The editor is driven by the same knowledge base that the language compiler uses to verify FIRST designs. The user is allowed to draw on a graphics screen using simple primitive operators that can be hierarchically built up using simple graphic instructions that can be found in any conventional drafting or drawing package.

Connections between the operators can be either "signals" or "controls", the exact number associated with each primitive being held in the knowledge base. The system allows for the following graphic operations

- place a primitive
- place an input pad
- place an output pad
- create a signal node
- create a control node
- connect primitives
- store a prescribed area as an object
- recall a stored object
- move,alter and delete primitives etc.

Plus the usual graphic operations

- pan
- zoom
- etc.

As the user builds up his representation errors can be detected as they occur. These errors are flagged with meaningful messages so that the user may understand the fault. The editor also functions in a mode that continuously prompts the user with help information depending upon the task being performed. An example of the basic information for the primitive "ADD" is outlined in Table 1.

More general information on this, or any other primitive can also be requested. The direct help information system can also be invoked from within the editor that allows the user to be guided through the complete hierarchy of help information from the completely general to the exacting detail.

Once the user is happy with his design he can then simulate the system and examine the results using the "SIMPLOT" program that acts like an oscilloscope, allowing the user to "poke" into the simulation program as it is running. This produces graphical output quickly rather than waiting for a complete simulation run to finish before examining the results, which could have been faulty from the start.

The simulation complete the user can use the interactive layout generator to produce the final chip plots and pattern generator tapes for mask commitment and final fabrication.

TABLE 1. Example of on-line help information the the primitive "ADD"

Primitive: ADD number of parameters = 4
signals in = 3, signals out = 2, ctls in = 1, ctls out = 0

parm	min val	max val	constraint	comments
1	1	32	none	latency of sum
2	0	1	none	predelay on addend 1
3	0	1	none	predelay on addend 2
4	0	1	none	predelay on carry in

AN EXAMPLE

The diagram in Figure 8 is a schematic of a complex to magnitude converter chip which uses a four-region approximation algorithm[5]. This was designed "by hand", the FIRST code entered using a text editor is shown in Table 2. The main point to make is that the designer would always draw a diagram before coding the circuit. On the diagram he would label the nodes to make it easier to code. The code would then be "compiled" and any errors corrected. The graphic input is shown in Figure 9 and the code that it generated is shown in Table 3.

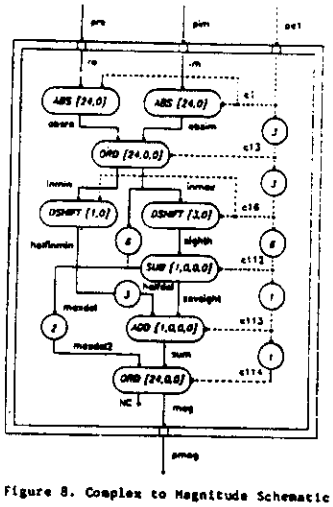


Figure 8. Complex to Magnitude Schematic

TABLE 2. Hand coded FIRST for the complex to magnitude chip.

```
CHIP CXTOMAG (pc1) pre, pim -> pmag
   SIGNAL absre, absim, inmax, inmin, eighth,
          halfinmin,seveight,halfdel, maxdel,
          maxdel2, sum, re, im, mag
   CONTROL c13, c16, c112, c113, c114, c1
   PADIN (pc1 -> c1) pre, pim -> re, im
   PADOUT mag -> pmag
   PADORDER VDD,pre,pim,pc1,GND,CLOCK,pmag
   ABSOLUTE [24,0] (c1) re -> absre
   ABSOLUTE [24,0] (c1) im -> absim
   ORDER [24,0,0] (c13) absre,absim -> inmax,
         inmin
   DSHIFT [3,0] (c16) inmax -> eighth
   DSHIFT [1,0] (c16) inmin -> halfinmin
   SUBTRACT [1,0,0,0] (c112) maxdel, eighth,
         GND -> seveight, NC
   ADD [1,0,0,0] (c113) seveight, halfdel,
         GND -> sum, NC
   BITDELAY [6] inmax -> maxdel
   BITDELAY [2] maxdel -> maxdel2
   BITDELAY [3] halfinmin -> halfdel
   ORDER [24,0,0] (c114) sum, maxdel2 ->
         mag, NC
   CBITDELAY [3] (c1 -> c13)
   CBITDELAY [3] (c1) -> c16)
   CBITDELAY [6] (c16 -> c112)
   CBITDELAY [1] (c112 -> c113)
   CBITDELAY [1] (c113 -> c114)
END
endofprogram
```

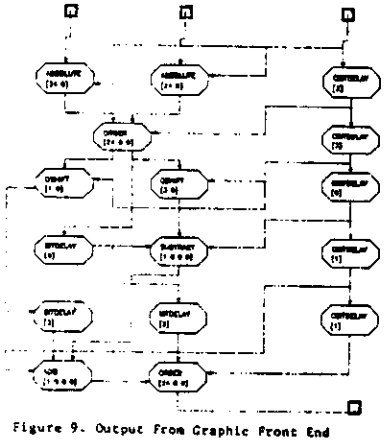


Figure 9. Output From Graphic Front End

TABLE 3. Computer generated FIRST for the complex to magnitude chip.

```
CHIP CXTOMAG (pc3) ps1,ps2 -> ps20
 SIGNAL s1,s4,s2,s5,s11,s12,s14,s15,s13,
 s16,s17,s19,s18,s20
 CONTROL c3,c6,c7,c8,c9,c10
 PADIN (pc3 -> c3) ps1,ps2 -> s1,s2
 PADOUT s20 -> ps20
 PADORDER VDD,pc3,GND,ps1,ps2,ps20,CLOCK
 ABSOLUTE[24,0](c3) s1 -> s4
 ABSOLUTE[24,0](c3) s2 -> s5
 CBITDELAY[3](c3 -> c6)
 ORDER[24,0,0](c6) s4,s5 -> s11,s12
 DSHIFT[1,0](c7) s11 -> s14
 DSHIFT[3,0](c7) s12 -> s15
 BITDELAY[6]s12 -> s13
 SUBTRACT[1,0,0,0](c8) s13,s15,GND -> s16,NC
 BITDELAY[3]s14 -> s17
 BITDELAY[2]s13 -> s19
 ADD[1,0,0,0](c9) s16,s17,GND -> s18 NC
 ORDER[24,0,0](c10) s18,s19 -> s20,NC
 CBITDELAY[3](c6 -> c7)
 CBITDELAY[6](c7 -> c8)
 CBITDELAY[1](c8 -> c9)
 CBITDELAY[1](c9 -> c10)
END
endofprogram
```

The differences between the two sets of data are slight. The node labelling is different and the order in which the primitives appears is different. This is dependent upon the order in which the diagram was entered into the graphic editor. The main point here is that the node labelling and decomposition to FIRST code has been automated. The need for the compilation cycle has also been removed as the FIRST code that is generated is guaranteed correct.

Modifications to the design after simulation also becomes simple. The graphic design is edited to incorporate any changes that are to be made, the FIRST code is automatically up to date with these changes, and again is guaranteed correct.

Finally a plot of the complex to magnitude chip is shown in Figure 10. This is by no means typical of a FIRST chip. The blank area on the layout is due to the fact that the chip is very small in the demonstration example that was used.

CONCLUSIONS

A method of graphic interaction to the FIRST silicon compiler has been demonstrated. The advantages of such a technique to designers over textual input has been outlined as well as the reduction in the design cycle time.

The expansion of the system to provide an integrated "silicon compiler workbench" is progressing with the "help" system already incorporated for on-line guidance. It is hoped that the simulation and layout programs will soon be enclosed within this system to provide an even better environment for the signal processing engineers to work in.

ACKNOWLEDGEMENTS

Thanks are due to our colleagues P.B. Denyer, D. Renshaw and K. Coplan. This work was carried out under SERC grant number GR/B/85418.

References

1. Denyer, P. B., D. Renshaw, and N. Bergmann, "A Silicon Compiler For VLSI Signal Processing," Proc ESSCIRC 82, Vrije Universitiet, Brussels, pp. 215-218 (September 1982).

2. Nash, J. H., "SKETCH --- A Two Dimensional Drawing Package," ISG Report ISG/82/09, (November 1982).

3. Ritchie, D. M. and K. Thompson, "The UNIX Time-Sharing System," Bell System Technical Journal, Vol. 57(6), pp. 1905-1929 (1978).

4. Kernighan, B. W. and D. M. Ritchie, The C Programming Language, Prentice-Hall (1978).

5. Filip, A. E., "A Baker's Dozen Magnitude Approximation and their Detection Statistics," IEEE Trans AES, Vol. AES-12, (1) pp. 87-89 (January 1976).

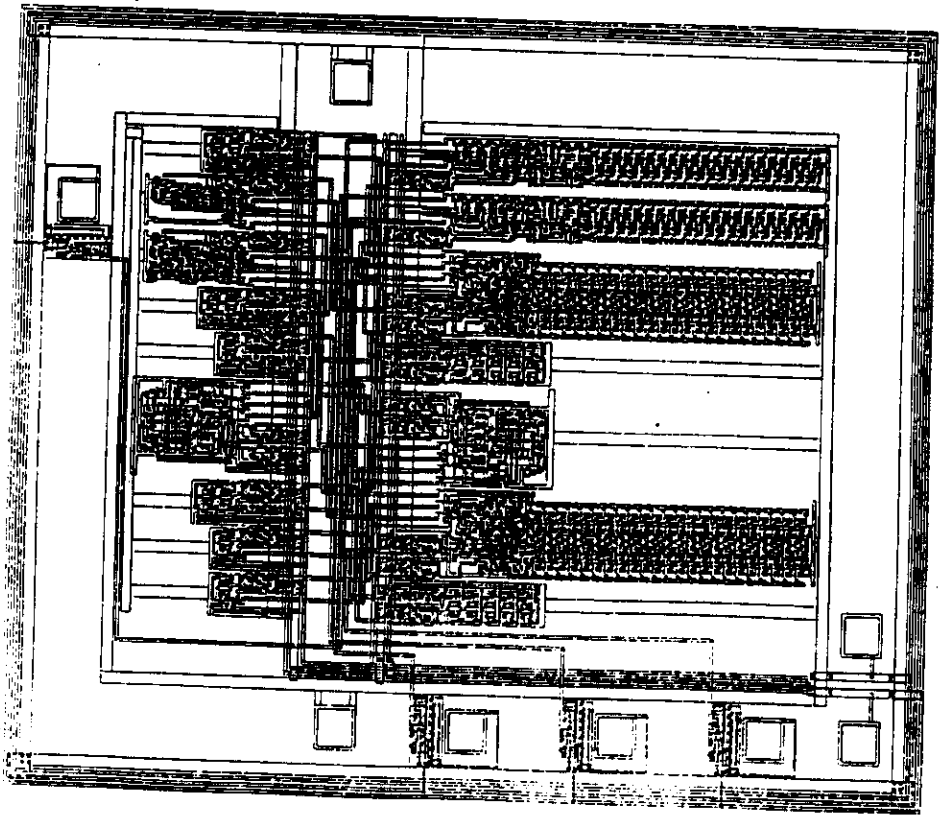


Figure 10. Complex to Magnitude Chip Plot

AS-4

VINCENT N. TRANS
EXAMINER
ART UNIT 234

QUALITY OF DESIGNS FROM AN AUTOMATIC LOGIC GENERATOR (ALERT)*

Theodore D. Friedman
and
Sih-Chin Yang

IBM Thomas J. Watson Research Center
Yorktown Heights, New York

7th DA Conf.
1970

Abstract

Automation of the design of computer logic has evoked wide-spread interest and activity. Nevertheless, detailed comparisons of automatically generated logic and manually prepared logic have not been made available.

The ALERT program is a logic generator which accepts as input a summary description of a new computer in a high-level language, and from this it compiles logic designs to carry out the functions specified. This paper examines the quality of logic generated by the ALERT system.

The specifications of several computers have been processed through the ALERT system. This report discusses the most comprehensive design processed, the central processor of the IBM 1800 computer.

The 1800 had been designed by conventional manual techniques prior to this study, and its logic schematics were, therefore, available for comparison with the logic generated by ALERT.

It was found that the automatically produced circuitry required 160% more components than used in the corresponding parts of the actual computer. Reasons for this discrepancy are considered, and methods are described which are expected to reduce the discrepancy between automatically generated designs and manual designs.

The study indicates that automatic generation of computer logic and circuitry from high-level system descriptions offers a practical and viable alternative to traditional methods of logic design. Within the limits of the study, the automatically prepared design was found to be correct, functional, and manufacturable.

Introduction

There has been wide interest in the possibility of automating the design of computer logic [1], [4], [5], [9], [11], [14], [16]. Possibility of an automated logic generator was demonstrated by Proctor in 1964 [12], however detailed comparisons of automatically generated logic and manually prepared logic have not been made available. The purpose of this paper is to present such a study.

The ALERT program, [7], [8], is a logic generator programmed to run on the IBM 7094. Logic generators such as ALERT should be distinguished from other design automation programs. Much of the development of new computers has been partially or fully automated, including wire routing, module placement, partitioning, circuit selection, checking, simulation, test generation, documentation and manufacturing [1], [3]. These automated systems, however, still rely on the human designer and engineer to prepare the logic design by hand before they can be used. The purpose of a logic generator is to automate preparation of the logic design itself.

As input to the ALERT program, the designer specifies the characteristics of the computer he desires, and from this the program generates logic equations which realize the specified features. These equations in turn can then be processed by standard Logic Automation [13] and Design Automation [3] programs to produce the circuitry and documentation of the desired computer. ALERT is analogous to a conventional compiler program because it translates its input from a high-level algorithmic form into a detailed machine-oriented form. But rather than generating a program as its output, ALERT generates logic designs for building hardware.

The input is expressed in a form of Iverson's APL notation [10]. To prepare the input, the user specifies the system characteristics of the desired machine, including the registers, memory size, word length or word mark conventions, instruction format and repertoire, indexing and other features. The behavior of the proposed machine is expressed by programs of micro-events.

It was necessary to develop special conventions to depict the machine organization and data flow information. APL had previously been used to describe the programming and functional characteristics of computer systems [6], but the descriptions did not specify the nature of the mechanisms used to provide these characteristics. To employ APL as a design specification language, the most important new convention imposed was for variables in the programs to be considered logic devices such as flip-flops, gate output lines, or registers, while the program statements themselves are interpreted as connections and gating among these devices. Accordingly, variables must be declared to be of fixed sizes and types. New statement types were introduced, such as assignment statements which affect only the set or reset lines of flip-flops (written: = 'SET', = 'RESET'). Because the input is punched on cards to be accepted by the 7094, APL symbols are transliterated. An-

* This report is a revision of IBM Research Report RC 2668.

RCL000158

other innovation in this version of APL is the period or statement terminator -- all entries to the right of a period in a card are considered to be comments.

The specifications of several computers were prepared in this way and were processed through the ALERT system. We will discuss the most comprehensive design processed, the central processor of the IBM 1800 computer.

The 1800 was selected because of its moderate size and its currentness. It is a general purpose machine intended for process control applications, and built with the IBM Solid Logic Technology circuit family. The 1800 had been designed and manufactured before this study was undertaken, and its logic schematics were available for comparison with the output logic generated by ALERT.

The 1800 Input Description

A listing of the input deck processed by ALERT describing the 1800 computer is shown in Appendix I.* This description was derived from the 1800 Systems Manual [15] and various engineering documents. An early description of the machine in Iverson's notation by K. Durgbacher was also helpful [2]. The description in Appendix I includes the major part of the 1800 Processor-Controller unit, but console devices, channels, input and output features, and the initial program loading facility are omitted. Twenty six of the 31 instructions in the machine's repertoire are included.

The description begins with declarations of registers, important flip-flops, and variables. The 1800's accumulator, for example, indicated in the first declaration statement by the name "A," is declared to consist of 16 flip-flops. The first occurrence of a name in the description fixes its size and dimensions. If no dimensions are given, as in the case of ADDFF in the second declaration statement, it is filed as a single bit.

Following the declarations, programs of microevents are listed. The first two programs are special pre-specified functions (Macro's), as indicated by a card with an "F" in column one. Macro functions are not entered into the design until later during processing when they may be "called" by other programs. Then they are copied and "plugged in" at each point of the design where a call occurred. The first Macro defines the design of decoders and the second defines a digital comparator.

Following these Macro functions are the ordinary programs, each of which is used to generate a section of the final machine. The first of these, MACO, specifies the memory access mechanism in the 1800. Following this are programs for interruption processing (INP), instruction counter advancement (ICNTR), interrupt queuing (LEVL), the adder (ADD), the shift control counter (SCCTR), register transfers (SHIFTL, SHIFTR, SWAP, DTOD, DTOA, ATOU, UTOA, XRTOSC, SCTOXR, XRTOA, and ATOXR). The next program (E) specifies the algorithms for execution of the instruction set, and the last program (CPU) describes the sequencing and control of the entire central processor.

Although space prevents review of the entire design, the reader may be assisted in understanding the computer description and the automatic design process by discussion of a part of the design. In particular, the Shift Control Counter program is examined in Appendices II and III.

Summary of the Findings

The 1800 description in Appendix I was processed by the ALERT system and by the IBM Logic and Design Automation systems to generate schematic logic diagrams of the machine. The diagrams were produced in a wholly automated fashion. One of these diagrams is shown and discussed in Appendix III.

The resulting design is considerably different in detail from the actual 1800 computer. Although the automatically produced design appeared for the most part correct and manufacturable, some systematic means were needed to evaluate it. The most scrupulous evaluation would have involved manufacturing the machine and then comparing its cost and performance with the real 1800. However such an approach would be excessively expensive. Moreover, a variety of flaws and inadequacies were discovered in the automatically produced design which would require correction before it could be built. (The flaws were generally minor and readily correctable, resulting from bugs in the ALERT programs or faults in the computer description. However, the 7094 computer used for running our study became unavailable before we could make those corrections.)

Rather than manufacturing the machine, therefore we felt that the next best method for evaluation would be to compare the numbers of components used in the real machine and in the automatically produced design. There were difficulties in this approach as well since the circuits implemented by the Logic Automation System differed from those used in the real 1800. Nevertheless, we made a circuit-by-circuit comparison of the two designs, checking the validity of the automatic design as we did so. The results are presented in Table I. Much of this table is concerned with various anomalies in the automatically produced designs, as described below.

Column 1 of the table identifies sections into which the 1800 design was partitioned. Partitioning was required because the design exceeded available memory in the 7094 computer and space limitations in the Logic Automation program. This partitioning, in turn, led to redundancies and incom-

* Some familiarity with the conventions of Iverson's APL Notation is desirable for proper understanding of this computer description. The reader is referred to the summary on pp. 199-203 of [6].

RCL000159

TABLE I

SUMMARY OF LOGIC BLOCK COUNTS FROM ALERT AND MANUAL DESIGNS

(1) Section of 1620 Design	(2) Number of Logic Blocks Produced by ALERT	(3) Number of Blocks after Regrouping for Circuitry	(4) Correction for Partitioning Effect	(5) Correction for Clocking Control	(6) Correction for Flip-Flops	(7) Corrected Total from ALERT	(8) Corresponding Sections of Real 1800	(9) Discrepancy of ALERT Total over Real 1800
Memory Access	347	424	-32	+32	+25	449	231	218
Interruption Processor	675	1004	-27	+19	+108	1109	525	199
Instruction Counter Control	128	239	---	---	+16	255	92	163
Interrupt Queuing	68	249	-17	---	---	122	62	60
Adder "	496	1173	-51	+49	+20	1171	521	650
Shift Control Counter	46	102	- 6	---	---	96	36	60
Register Processor	1065	1750	-184	---	+80	1646	587	1059
Instruction Execution	941	1598	-198	+285	---	1685	554	1131
Processor Control	934	1329	-207	+115	+18	1305	484	1021
TOTAL	4372	8018	-718	+468	+267	8033	3072	4961

TABLE II

SUMMARY OF DISCREPANCIES

	Corrected output from ALERT	8033 logic blocks
	Size of corresponding sections of real 1800	3072
	Discrepancy of ALERT over real design	4961 or 160% more blocks from ALERT than in real design
(minus)	Estimated reduction by modifying circuit implementation rules	2357
(minus)	Estimated Reduction by revising input description	568
(minus)	Estimated reduction by system improvements	1020
	Theoretical Discrepancy	1016, or 33% more blocks from ALERT than in real design

RCL000160

place coordination of the sections of the design, which are discussed later.

Column 2 lists the number of logic blocks produced by ALERT for each section. A logic block is a circuit which performs a logical function such as an AND gate, an OR gate, or an inverter. The output of ALERT represents idealized logic blocks, since circuit restrictions were not considered in producing these results. A total of 4372 idealized logic blocks were used in the design.

Column 3 lists the number of logic blocks which resulted after the idealized logic was automatically converted into actual circuitry by the IBM Logic Automation and Design Automation programs. In this conversion the designs from ALERT were reprocessed to cause them to conform to circuit constraints such as fan-in, fan-out, and module pin restrictions. This conversion caused extra logic blocks to be added into design to satisfy the engineering constraints: The total logic block count was increased from 4372 to 8018.

Column 4 shows the effects on block counts attributed to partitioning of the Design File.

The partitions segregated parts of the design that were functionally related. This led to the following anomalies:

(a) It was impossible to combine logic gates which have identical inputs if they appear in different sections.

(b) Redundant inverters were sometimes used. For example, suppose a signal, X, and its complement, $\overline{X}$, are available in the Adder Section, but only the X signal is represented in the CPU section. If $\overline{X}$ is also needed in the CPU a new inverter will automatically be generated instead of obtaining the complemented signal from the Adder. Redundancy also occurs in the case of extra gates supplied for driving loads.

(c) OR gates located in distinct sections and which have the same output destinations were not combined.

The figures in Column 4 were derived manually by counting instances of these anomalies.

The partitioning effects could be reduced or eliminated if larger storage capacity were available for ALERT and if storage restrictions in the Logic Automation program could be relieved.

Column 5 of Table I reports counts of logic blocks required for phase counters and their decoding networks. The decoded outputs are used to form timing control signals. In the ALERT system, the sizes of phase counters are determined automatically, however the logic elements needed to construct them were omitted from the final output. To correct this fault, these missing elements were manually added to the total logic count.

It may be noted that certain sections have no entry in Column 5. These sections correspond to programs with manually specified control signals which are generated elsewhere — these sections consequently do not require phase counters. A total of 480 logic blocks are estimated for the phase counters.

Column 6 lists the number of flip-flops required. The Logic Automation program cannot represent flip-flops unless its input data is specified in Conjective Word Format [13]. Since the results from ALERT were not produced in that format, flip-flops were not represented in the output logic drawings and it was necessary to supply them manually.

A given flip-flop may be used by several sections, but it is counted only at its initial use.

Column 7 shows the block counts from ALERT corrected for the partitioning effect, clocking control and flip-flops. This represents the sum of columns 3, 4, 5, and 6, providing a total of 8033 blocks.

Column 8 indicates the component counts used in the corresponding sections of the real IBM 1800. These figures were obtained by examination of the 1800 engineering manuals. A total of 3072 logic blocks were counted.

Column 9 indicates the discrepancy in logic block count from ALERT in column 7 over the real design. A total discrepancy of 4961 logic blocks was found, that is, 160% more logic blocks were generated by ALERT than were used in corresponding sections of the manual design.

Projected Improvements

The discrepancy between ALERT and the actual design can be reduced by the following methods, summarized in Table II.

(a) Extensions of the Choice of Circuit Types in the Logic Automation Programs

The ALERT logic equations were implemented using only three types of circuits, namely, AI (AND Inverter), AOI (AND-OR Inverter) and N (Single Input Inverter with driving power). These contrast with dozens of circuit types used in the actual design. Because of this, more logic blocks were generated than would be if there were a greater choice of circuits. This situation is compounded by the partitioning effect as well as by input and system deficiencies (see below). A total of 2357 logic blocks are attributed to this cause.

(b) Revision of the 1800 Description

Examination of the output logic indicated that if certain input statements were respecified or regrouped, the number of logic blocks required could be reduced without changing the intended functions. For example, saving may

RCL000161

sometimes be achieved if a register is pre-cleared before data is gated into it. Saving would also result if common signals would be combined as illustrated in Figure 1. In that case, the four inverters of the input signal "3" in Figure 1(a) could be replaced by one input in Figure 1(b). This saving, however, depends on the types of circuits available for implementing the logic. Another saving could be realized by rearranging input statements to reduce the number of timing phases in a program. There are several other techniques as well that would result in savings. It is estimated that 568 logic blocks can be eliminated from the output if all these input revisions would be incorporated.

(c) System Improvements

A number of improvements in the ALERT system have been projected although they have not been implemented. One improvement is sharing phase counters. Several programs could share a common counter to reduce the number of counters required. Also, within a single program, statements may be divided into groups and the phase counter will be reset to one at the beginning of each group. This approach would reduce the size of the counter and the number of its decoded outputs. However, more control signals would then be needed to distinguish each group.

Another possible improvement would involve greater coordination of array inputs. For example, assume A, B, C, D are all 4 bit vectors. The statement

```
A = B 'AND' C 'AND' D
```

will presently produce the gating shown in Figure 2(a). The improved version should produce the structure shown in Figure 2(b).

Still another improvement would be to cause ALERT to generate output in the Injective Word Format used in the Logic Automation System. This would avoid difficulties such as the limited circuit choice. Various other improvements are also projected, but are too detailed to list here. These improvements, it is estimated, would save a total of 1020 logic blocks in the 1800 design.

After all corrections, the automatically generated logic would then exceed the manual design by 1016 blocks. (See Table II.)

Conclusions

The current study suggests that automatic generation of computer logic and circuitry from high-level system descriptions offers a practical and viable alternative to traditional methods of logic design. Apart from a number of remediable errors, the automatically prepared design was found to be correct, functional, and manufacturable. Although the automatically produced circuitry required more than twice as many components as used in the corresponding manual designs, this discrepancy is expected to be reduced by projected system improvements.

It is anticipated that compilers of computer logic will provide some of the benefits that conventional compilers have provided to programmers. Automated techniques should not only aid development directly, but should facilitate checking, rework and documentation.

RCL000162

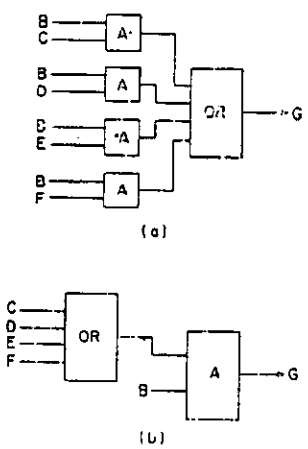
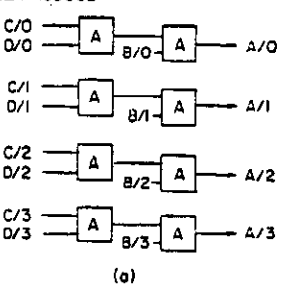


Fig. 1. Combining common signals.

GIVEN A, B, C, D, ALL 4 BIT VECTORS; THEN
A = B 'AND' C 'AND' D
WILL PRODUCE



THE IMPROVED VERSION SHOULD PRODUCE

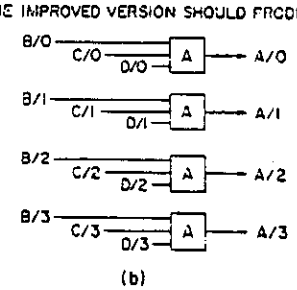
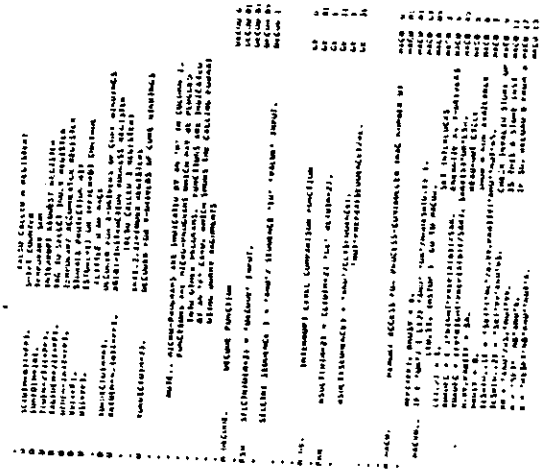
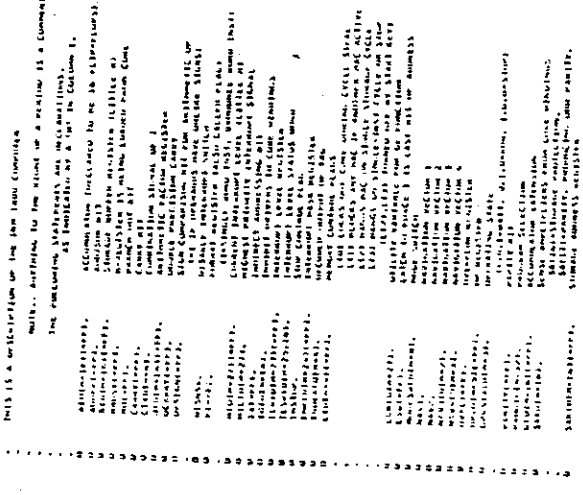
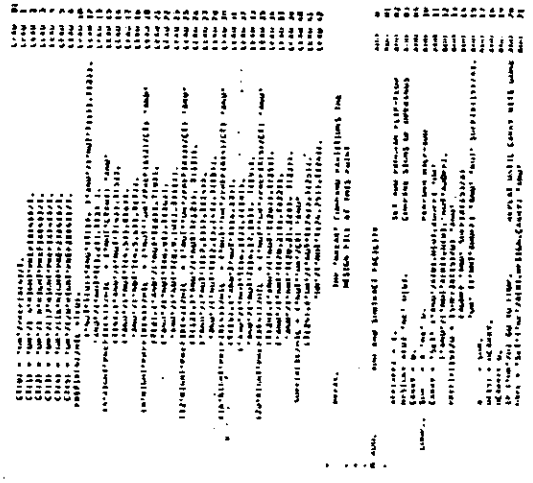


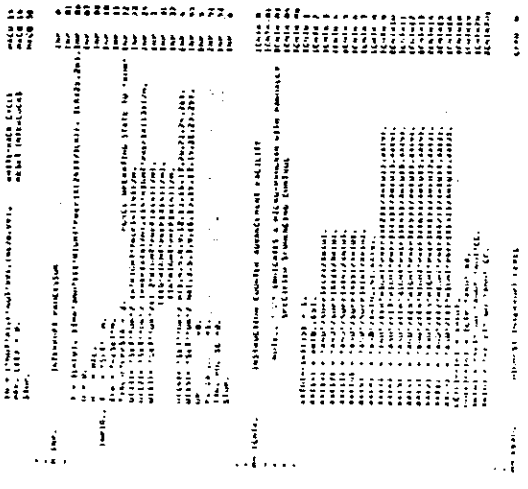
Fig. 2. Coordination of array signals.

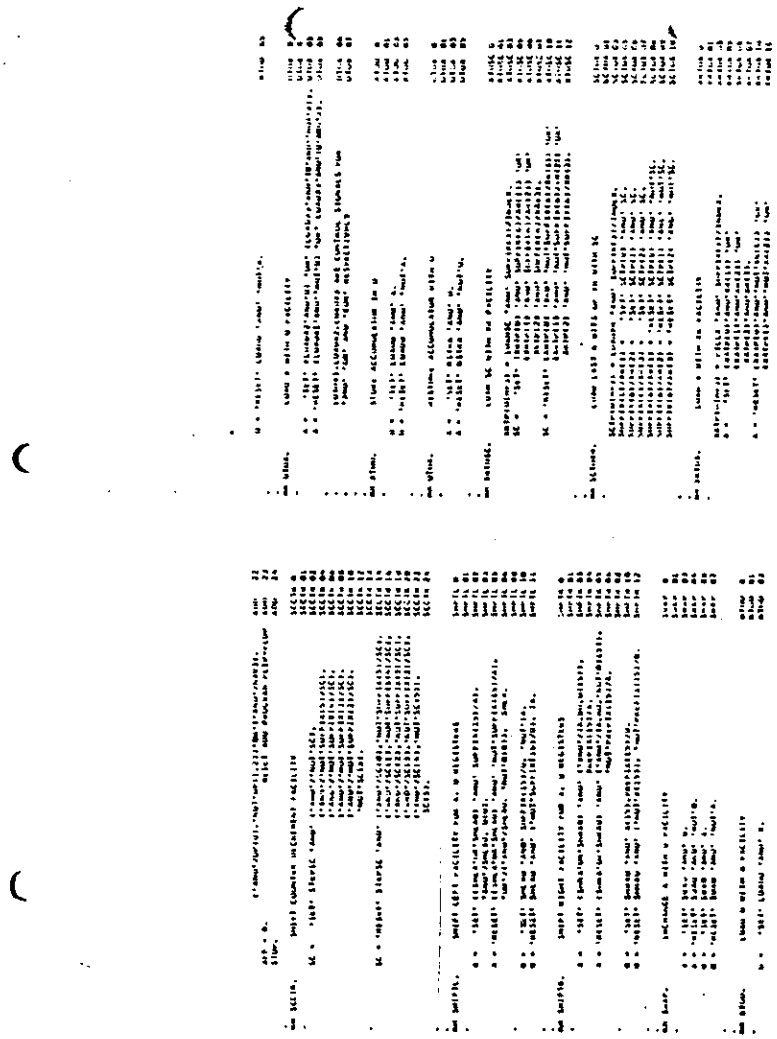


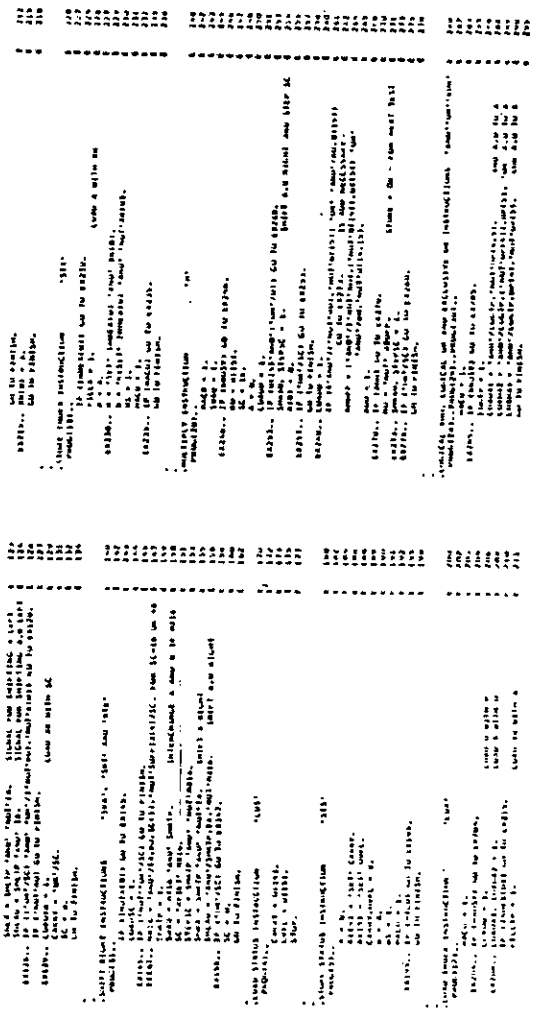
APPENDIX I

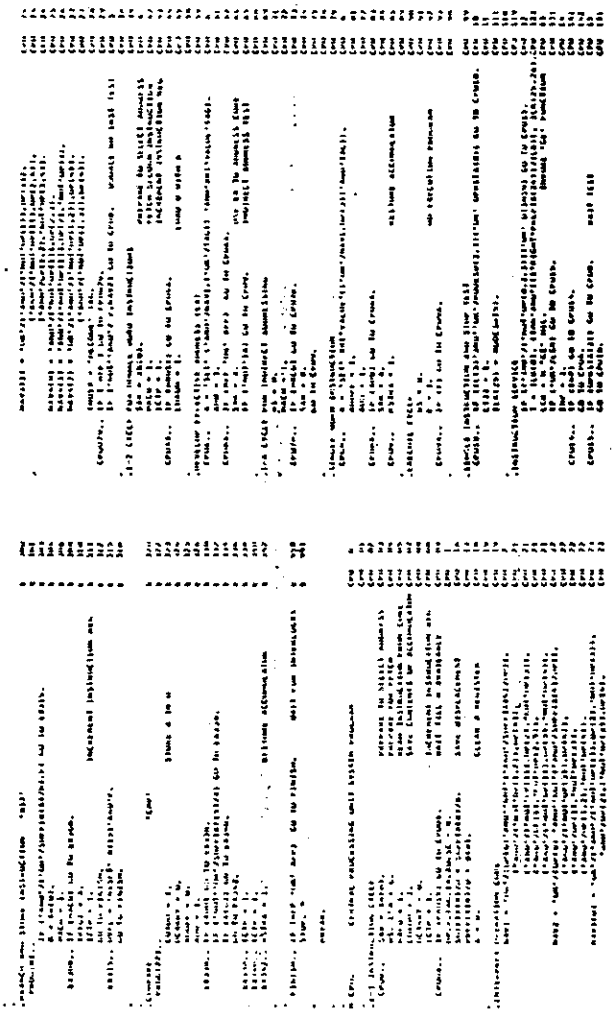












Appendix II

The 1800 performs left and right shifts of its A register (or of the A and Q registers together for extended shifts). The shift instruction algorithms are represented in programs 2 and 3 of the instruction execution ("E") program. According to these algorithms, the shift count is loaded into the Shift Control Counter, SC, and then a loop is executed. During every iteration of the shift loop a signal (SHLA, SHLAQ, SHRA, or SHRAQ) is activated which causes a one-bit shift to be accomplished by the SHIFTL or SHIFTR program. Every iteration of the loop also activates a variable, STEPSC, which decrements the bit count in SC. The shift loop terminates when SC equals zero. Thus, the SC register controls the number of positions which the accumulator is shifted.

The Shift Control Counter program (SCCTR) indicates how the STEPSC signal is used to decrement the SC register. Since this program is particularly simple and straightforward, it will be used to illustrate the process of automatic design. This program is shown in Figure 3 as it appears in the 1800 description.

The "NM" heading in the first card of this program identifies it as a Micro-program with Manually specified control. This prevents ALERT from generating sequence control logic. Instead, this section depends for control only upon the signals explicitly designated in the program.

The program consists of just two assignment statements. The first statement assigns signals to the set side of the SC register flip-flops, and the second assigns signals to the reset side. These signals, which are Boolean functions of the current state of the SC bits, are inhibited except when STEPSC = 1. Since ALERT provides no extra control logic in this program, an assignment will occur whenever STEPSC is pulsed. At such an occurrence SC will acquire a new value, and this value -- as the reader may verify -- will constitute a decrease by one of the binary number represented in SC previously.

When the input deck was processed by the ALERT processor, the Shift Control Counter program was translated into the logic equations shown in the print-out in Figure 4. The original assignment statements were decomposed into elementary statements, each of which defines a single gating function and identifies signal sources and destinations. For example, the first equation specifies that the complement of bits 0, 1, 2, 3, 4, and 5 of SC are ANDed together and the resultant variable is designated as "T59." Such "T" variables are intermediate elements supplied automatically during translation of the source statements. This term corresponds to the sub-expression,

```
" ('AND'/'NOT'SC) ",
```

in the first assignment statement of the source document (Figure 3). During further processing of the 1800 design, variables representing signals from other sections of the design were combined with equations in the SCCTR section. The resulting equations indicate need of 14 two-input AND gates, two 3-input ANDs, two 4-input ANDs, two 5-input ANDs, two 6-input ANDs, 12 one-input OR gates, and 12 two-input OR gates. Thus a total of 46 elementary logic gates, or "blocks," are required.

When these equations were processed by the IBM Logic Automation programs, the 12 one-input OR gates were discovered to be redundant, and the revised block count dropped to 34.

The designs were processed further in order to comply with engineering and circuitry constraints of Solid Logic Technology. Extra gating was automatically provided and the block count then rose to 102. Six pages of logic schematics were automatically drawn from the resultant design of the SCCTR section. The first of these pages is shown and discussed in Appendix III.

The design of the SCCTR section as generated by ALERT is shown in summary form in Figure 5. The design of the corresponding section of the actual machine is represented in Figure 6. For simplicity, in both these figures special gating imposed by the circuit connection rules has been omitted, and signals and gates not related to the shift counter decrement process also are not shown. In the real 1800 the decrement signal (here called "DESC") is gated only to the least significant bit (SC/1) which will toggle when a pulse is applied (see Figure 6).* If SC/1 is in the "0" state, changing it to "1" will cause toggling of the next higher bit, SC/2. Likewise every SC bit will, when toggled, itself toggle the next higher bit. This chain reaction may propagate to the highest bit of the counter, but will cease at a bit with an original value of 1. This process causes the value represented in SC to be decremented.

The design generated by ALERT employs a different method of counting. This difference, however, does not result from the ALERT system itself but is due to the way the counting function had originally been specified in the APL description. In the design produced automatically, the counter is organized so that the decrement signal is applied to all six bits of SC simultaneously rather than by a bit-by-bit propagation mechanism.

The logic generated by ALERT used AND gates having from two to seven inputs each (Figure 5). However, after automatic revision to meet circuit needs, there was an increase in gates required. For example, in terms of the idealized logic generated by ALERT, two gates were used to provide the signal to set the most significant bit of SC

* Subscripting conventions differed in the output generated by ALERT and in the engineering documents of the real machine. In the ALERT output, the six bits of SC are denoted, from most significant to least, as SC/0, SC/1, ..., SC/5. In the real 1800, these bits are denoted, respectively, as SC/32, SC/16, ..., SC/1.

(Figure 7(a)). When this design was converted into circuitry, 13 gates were required, as shown in Figure 7(b). This section of design is also shown in final printed form in Appendix III. This discrepancy between the automatically produced design and the manual design is due mainly to restrictions on the available circuit types. Because inversion takes place for all circuits used, to get the proper polarity, extra logic levels were needed. This effect is aggravated by fan-in restrictions and circuit card layout considerations. Also, because of the partitionality of design, signals generated in one section may not be available in another section. This is the case for $\overline{SC/0}$ and $\overline{SC/1}$. Inverters are used to generate these complement signals from SC/0 and SC/1 instead of obtaining them directly from another section. For the entire shift counter, a total of 102 logic elements are generated by ALERT as against 36 for the real 1800. If the projected improvements are made in ALERT, the count of the logic blocks generated is expected to be reduced to 42.

```
NM SCCTR,       SHIFT COUNTER DECREMENT FACILITY

        SC =    'SET' STEPSC 'AND' ('AND'/'NOT'SC),
                                   ('AND'/'NOT'SUPPIRIS1/SC),
                                   ('AND'/'NOT'SUPPIRIA2/SC),
                                   ('AND'/'NOT'SUPPIRIB3/SC),
                                   ('AND'/'NOT'SUPPIRIZ0/SC),
                                   'NOT'SC(9).

        SC = 'RESET' STEPSC 'AND' ('AND'/SC(0),'NOT'SUPPIRIS5/SC),
                                   ('AND'/SC(1),'NOT'SUPPIRIA4/SC),
                                   ('AND'/SC(2),'NOT'SUPPIRIB3/SC),
                                   ('AND'/SC(3),'NOT'SUPPIRIZ2/SC),
                                   ('AND'/SC(4),'NOT'SC(5)),
                                    SC(9).
```

Fig. 3. The Shift Control Counter program.

```
SCCTR 0
SCCTR 01
SCCTR 02
SCCTR 04
SCCTR 08
SCCTR 10
SCCTR 12
SCCTR 13
SCCTR 14
SCCTR 18
SCCTR 20
SCCTR 22
SCCTR 24

EQUATIONS
T5V0=SC/0*~SC/1*~SC/2*~SC/3*~SC/4*~SC/5
T4A0=SC/1*~SC/2*~SC/3*~SC/4*~SC/5
TA1*~SC/2*~SC/3*~SC/4*~SC/5
TA2*~SC/3*~SC/4*~SC/5
TA3*~SC/4*~SC/5
TA4/0=STEPSC*T159
TA4/1=STEPSC*T4U
TA4/2=STEPSC*T61
TA4/3=STEPSC*T62
TA4/4=STEPSC*T63
TA4/5=STEPSC*~SC/5
T514/0=TA4/0+T112/0
T514/1=TA4/1+T112/1
T514/2=TA4/2+T112/2
T514/3=TA4/3+T112/3
T514/4=TA4/4+T112/4
T514/5=(A4/5+T112/5
SC/5/0=T514/0
SC/5/1=T514/1
SC/5/2=T514/2
SC/5/3=T514/3
SC/5/4=T514/4
SC/5/5=T514/5
T65A*SC/0*~SC/1*~SC/2*~SC/3*~SC/4*~SC/5
TAA*SC/1*~SC/2*~SC/3*~SC/4*~SC/5
TA7*SC/2*~SC/3*~SC/4*~SC/5
TAA*SC/3*~SC/4*~SC/5
TAA*SC/4*~SC/5
T7D/0=STEPSC*165
T7D/1=STEPSC*166
T7D/2=STEPSC*167
T7D/3=STEPSC*168
T7D/4=STEPSC*169
T7D/5=STEPSC*SC/5
T515/0=T7D/0+T117/0
T515/1=T7D/1+T117/1
T515/2=T7D/2+T117/2
T515/3=T7D/3+T117/3
T515/4=T7D/4+T117/4
T515/5=T7D/5+T117/5
SC/R/0=T515/0
SC/R/1=T515/1
SC/R/2=T515/2
SC/R/3=T515/3
SC/R/4=T515/4
SC/R/5=T515/5
END OF EQUATIONS
```

Fig. 4. Print-out of Boolean equations generated for the Shift Control Counter.

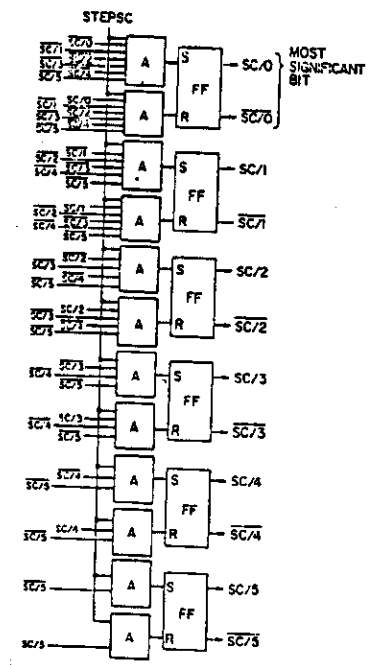


Fig. 5. Summary of the automatically produced design of the Shift Control Counter.

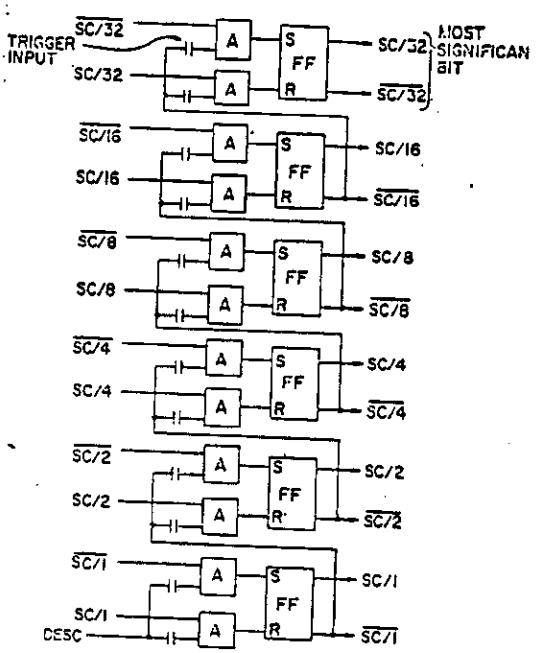


Fig. 6. Summary of the actual design of the Shift Control Counter.

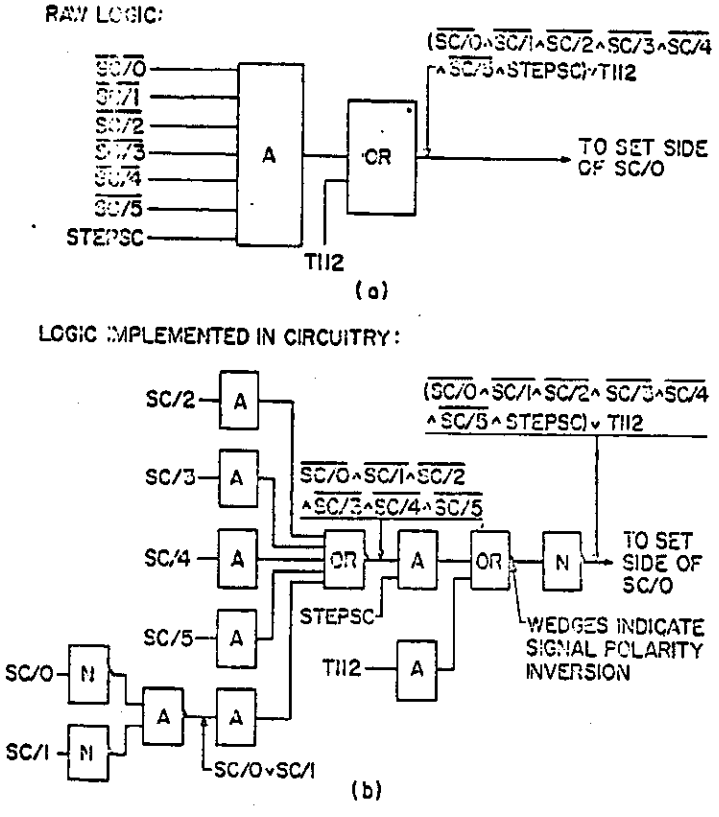


Fig. 7. Idealized logic and its implementation in circuit blocks.

Six pages of schematic logic diagrams were produced automatically from the designs of the Shift Control Counter section. The first diagram is shown in Figure 8. This diagram represents the logic to provide the signal, SC/S/0, to set bit 0 of the SC register.

A large proportion of the resulting gates serve only to satisfy circuitry requirements rather than to carry out logical functions. It is not possible to investigate the quality of design without consideration of circuit connection rules. But the actual rules differ from one family of circuitry to another and are not part of the logic generation processes of ALERT. Although circuit connection problems are mentioned in this section, these topics cannot be adequately covered here.

At the lower left, two N blocks (single-input inverters) are used to generate the $\overline{SC/0}$ and $\overline{SC/1}$ signals from SC/0 and SC/1. (Due to formatting conventions, the slash in these names is not printed on the drawings.). Since $\overline{SC/0}$ and $\overline{SC/1}$ are also needed in other drawings, two lines are shown tapping off from them. $\overline{SC/0}$ and $\overline{SC/1}$ are used as inputs to an AND inverter, the output of which provides the function SC/0 ∨ SC/1. This gate is used to reduce the number of input lines to the OR inverter at the center of the drawing. Five single-input AND gates feed this OR inverter. (These ANDs do not serve a logic function but are required for circuit needs.) The inputs to the five AND gates are SC/2, SC/3, SC/4, SC/5 and the combined signal SC/0 ∨ SC/1. Thus the output of the OR inverter becomes $\overline{SC/0} \wedge \overline{SC/1} \wedge \overline{SC/2} \wedge \overline{SC/3} \wedge \overline{SC/4} \wedge \overline{SC/5}$. This output is then ANDed with the stepping pulse STEPSC, and the result defines the set side of SC/0. However, another signal, T112, not related to counting, is also needed as a set input. To enable them to use a single line to SC/0, the stepping control and the T112 signals are ORed together. Since the circuit used for the OR function also inverts the output, an extra N inverter is used to restore it to correct polarity. The final result (SC/S/0) is thus

$$(\overline{SC/0} \wedge \overline{SC/1} \wedge \overline{SC/2} \wedge \overline{SC/3} \wedge \overline{SC/4} \wedge \overline{SC/5} \wedge STEPSC) \vee T112.$$

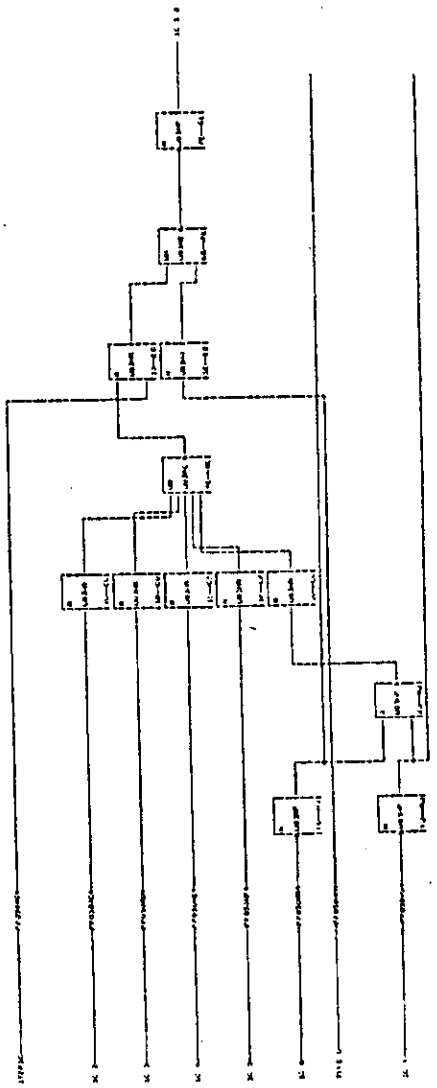


Fig. 8. A schematic logic diagram produced automatically from output from ALERT

RCL000175

Acknowledgment

Basic approaches used in this system were originally proposed by C. J. Coleman, Los Angeles Scientific Center.

References

1. M. A. Breuer, "General survey of design automation of digital computers," Proceedings of the IEEE, Vol. 54, pp. 1708-1721, December 1966.

2. J. Burgbacher, "Processor-controller of IBM 360 system, a formal description," IBM Technical Note TN 02.559.01, July 1966.

3. D. W. Case, M. H. Graf, L. E. Griffith, A. R. Marcq, W. B. Murley, and T. H. Spence, "Solid logic design automation for IBM System/360," IBM J. Res. and Dev., Vol. 8, pp. 127-140, April 1964.

4. T. Chu, "An ALGOL-like computer design language," Communications of the ACM, Vol. 8, pp. 607-615, October 1965.

5. I. Estrin and R. Mandell, "Metacompiler as a design automation tool," 1966 Proc. Share Design Automation Conference.

6. D. Falkoff, K. E. Iverson and E. H. Sussenguth, "Formal description of System/360," IBM Syst. J., Vol. 3, No. 2, pp. 198-262, 1964.

7. T. D. Friedman and S. C. Yang, "Methods used in an automatic logic design generator (ALERT)," IEEE Trans. on Computers, Vol. C-18, No. 7, pp. 593-614, July 1969.

8. _____, ALERT: A program to compile designs of new computers," Digest 1st Annual IEEE Computer Conference, pp. 128-130, September 1967.

9. R. F. Gorman and J. P. Anderson, "A logic design translator," 1962 Proc. FJCC, pp. 86-96.

10. K. E. Iverson, A Programming Language, New York: Wiley, 1962.

11. R. Metze and S. Seshu, "Computer compiler of I -- Preliminary report," Coordinated Science Lab., University of Illinois, Urbana, Rpt. R-303, August 1965.

12. R. Proctor, "A logic design translator experiment demonstrating relationship of language systems and logic design," IEEE Trans. on Electronic Computers, Vol. EC-13, pp. 422-430, Aug. 1964.

13. T. Roth, "Systematic design of automata," 1967 Proc. FJCC, pp. 1093-1099.

14. E. Schlaeppi, "A formal language for describing machine logic, timing and sequencing specifications," IEEE Trans. on Electronic Computers, Vol. EC-13, pp. 439-448, August 1964.

15. "IBM 1800 Functional Characteristics," IBM System Reference Library, Form A26-5913.

16. W. A. Wiley and D. L. Dietmeyer, "Translation of a DDL Digital System Specification to Boolean Equations," IEEE Trans. on Computers, Vol. C-18, No. 4, pp. 305-313, April 1969.

RCL000176

AR-6

VINCENT N. TRANS
EXAMINER
ART UNIT 234

17th DA Conference 19S

A New Look at Logic Synthesis

John A. Darringer
William H. Joyner, Jr.

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Abstract

Despite many attempts to generate hardware implementations automatically from functional specifications, the literature does not record any commercial success. Previous efforts have dealt primarily with technology-independent primitives and have emphasized circuit minimization. However, larger scales of integration have made other design requirements and technology restrictions as important as circuit count, and have increased the cost of making an engineering change. Thus it is becoming increasingly important to insure that initial chip designs are correct. This paper outlines an investigation into the feasibility of logic synthesis in this new context. A system is described which will produce a naive implementation automatically from a functional specification, and then will interact with the designer, allowing him to evaluate it with respect to these many factors, and to improve it incrementally by applying local transformations until it is acceptable for manufacture. The use of simple local transformations will insure correct implementations, will isolate technology-specific data, and will allow the total process to be applied to larger VLSI designs. This approach has been tested on the design of a single chip with encouraging results. A prototype synthesis system is now being used to perform further experiments.

Logic Synthesis

The goal of logic synthesis is to accept functional specifications for a hardware unit and to generate automatically a detailed, technology-specific implementation comparable in quality to that of an experienced engineer. The nature of this problem depends on the level of the functional description, the set of implementation primitives, and the criteria of acceptability. Initially, we are examining a restricted form of synthesis. That is, we are concentrating on the problem of generating masterslice implementations from low-level register transfer specifications in which all memory elements are specified. The combinational network between the memory elements must be generated and the entire function must be implemented with primitives from a specified set. Further, the implementation must satisfy given performance requirements and technology restrictions. We hope to determine the feasibility of interactive synthesis in this context, and the applicability of techniques of program analysis and optimization.

There has been much work on automating logic design and, although many effective tools have been developed to aid the implementer, synthesis is not considered commercially successful. Early work centered on developing algorithms for translating a boolean function into a minimum two-level network of boolean primitives. Extensions were developed for handling limited circuit fan-in and alternative cost functions [Bre72, Die71]. Because these algorithms search for minimal implementations they are necessarily exponential and require too much space and time to be used on most actual designs.

Later efforts attempted to raise the level of specification. The DDL work at Wisconsin [Dul68, Die71], APDL at Carnegie-Mellon University [Dar69], and ALERT at IBM [Fri68] all began with behavioral specifications and produced technology-independent implementations in the form of boolean equations. The results were usually more expensive than manual implementations and did not take advantage of the target technology. For example, the ALERT system was validated on an existing design, the IBM 1800, and the implementation produced required 160% more circuits than the manual design [Fri70].

In attempts to generate more efficient logic and to give the user more control over the implementation, other strategies were tried [Sch62, Mes68, Hil73]. These constrain the specification language so that there is nearly a one-to-one correspondence between the specification and the implementation. Of course, this constraint also decreases the advantage provided by the system.

The RT-CAD project at Carnegie-Mellon [Sie76] has developed tools to support the total machine design process from architectural design to physical layout. So far, some tools have been demonstrated for the early part of the design cycle, but the total system is not yet complete [Bar73, Tho77, Sno78, Haf78]. There is also an effort in Japan to develop a system to help a designer translate an existing SSI or MSI implementation into LSI [Nak78].

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1980 ACM 0-89791-020-6/80/0600/0543 $00.75

RCL000177

A New Approach

The system we propose is not intended to be a completely automatic replacement for the manual design process. Instead, it is an interactive system in which the user operates on a logic design at several levels of abstraction. He can simplify an implementation at one level, and when satisfied can move to the next level. He does this by applying transformations, either locally or globally, to achieve the simplification or refinement. By being able to operate on the implementation at several levels, the user can often make a change at one level that will cause a great simplification at a lower level.

Our approach differs from previous efforts in other respects. We see the logic synthesis problem as one of finding a feasible (not an optimal) implementation: a network of primitive boxes that satisfies a large number of constraints. These include timing constraints, a restricted library of primitives, driver requirements, clock distribution rules, fan-in and fan-out constraints, rules for testability (LSSD), and switching currents, in addition to limits on gate count and I/O pins. Though previous attempts have considered some of these factors, most have been limited to technology-independent transformations and have not given full consideration to all the requirements and restrictions placed on the logic designer today.

We are also attempting to limit our transformations to local changes that do not require exponential time or space. If we are successful then our approach may scale up to handle larger VLSI chips. Previous efforts have relied on algorithms that do not scale to realistic problems.

In designing a system to apply these transformations, we have been influenced by the design of optimizing compilers that translate a higher level description into a low level implementation through several intermediate levels, which are convenient for analysis and optimization. Similarly, we first convert a specification in a hardware description language to a network of typed boxes: ANDs, ORs, NOTs, memories, parity checkers, etc. We then allow transforms to be applied to this network to move the design closer to one that can be implemented in the available primitives of the target technology.

Our initial task is to identify the factors that influence a designer, to develop local transformations for altering these factors, and to learn how to apply these transforms (globally or at specific points) to generate implementations in an actual target technology. If these efforts are successful we could expect shorter hardware development times, lower development costs, and fewer errors since we will show that the transformations preserve the specified function.

A System for Logic Synthesis

The principal input to a synthesis system is a functional description of the logic to be implemented. We accept this description in a language similar in level to CDL [Chu65], and transform it into a network of boolean functions, memories, inputs, and outputs in a straightforward manner. Initially, we are considering functions implemented by a single chip. Therefore, we require an interface description giving the polarities of the inputs available and the outputs required, and the sources and destinations of inputs and outputs -- information needed to assign receivers and drivers. Information is also needed about the target technology and its design rules, describing the primitives available, the fan-in and fan-out requirements, and the timing constraints. This information is held in a technology file.

The analysis involved in logic synthesis takes place at several levels of abstraction. Transformations such as identification of common sub-expressions will be appropriate at every level, while simultaneous switching could be more conveniently analyzed at a high level (where registers and decoders are easily recognizable) and timing analysis at a lower level (where the real hardware primitives are available). At each level the description will be in terms of primitives of that level, such as registers, shifters, and decoders or at a lower level shift register latches, and-inverts, and dot-ands. These primitives will have two interpretations: a set of properties convenient for analysis, and an implementation (possibly parametric) in terms of lower level primitives. This is in contrast to a "macro" approach where the only meaning of a box is its expansion in terms on the lowest level primitives.

Figure 1 shows the organization of the logic synthesis system. The inner box delimits the concern of this paper. Its inputs are the register transfer specification, the interface constraints, and the design rules which apply to the target technology. The output is a detailed implementation in terms of the primitives of the target technology, which is submitted to placement and wiring. Though gate and pin restrictions and rough timing constraints should have been enforced by the transforms, some violations of these may be apparent only after placement and wiring are complete. The loop back into the synthesis system permits other attempts at synthesis, either with adjusted design rules and technology parameters or with different user input, until a wirable implementation is achieved.

RCL000178

An Experiment

To investigate the feasibility of the approach to synthesis outlined above, we conducted an experiment with a chip of an existing processor. This chip had been specified at the functional level and then implemented by engineers. The existence of the engineers' implementation of the chip allowed us to compare the two implementations and to study the differences. The transforms of the experiment were performed initially by hand and later by our prototype system, as a first step in learning how logic design in this context is done and what parts of it are amenable to automatic methods.

The initial description used as input to this experiment was similar to a CDL [Chu65] description. It specified the function that was to be implemented by a single chip, the function's inputs and outputs, and, importantly, each memory element that is to appear in the final implementation. For generating a technology-specific implementation, moreover, additional information about the inputs and outputs will be needed: polarity of inputs available, polarity of outputs re-

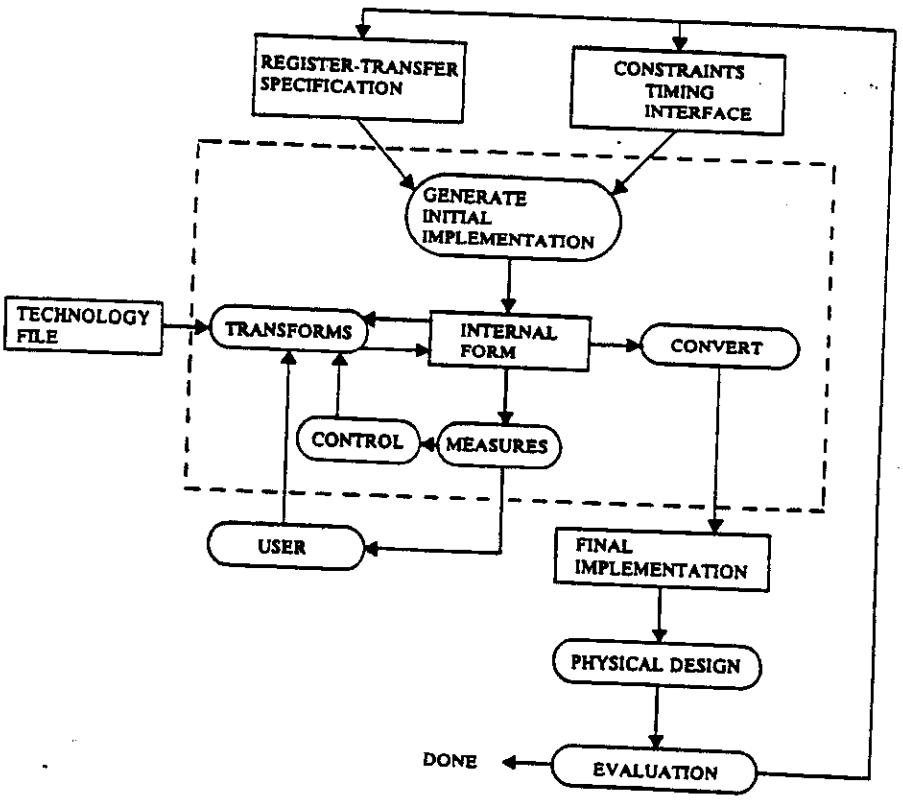


Figure 1. The Logic Synthesis System

RCL000179

quired, source of inputs and destination of outputs (for receiver and driver requirements), and frequency of change of inputs (to determine timing requirements). This information is part of an interface specification separate from the functional description and used later in the sequence of transformations.

Also required for the synthesis system is information about the target technology, either in a technology file or built into the transformations. In this experiment a logic chip is a masterslice design with 96 I/O pins and 704 sites, equally divided between 3- and 4-input NAND gates. The designer, however, also uses macro-like definitions of higher level boxes made up of these NAND gates. In addition, the technology file contains restrictions on the use of the primitives and macros, fan-in and fan-out requirements, timing constraints, and clocking and powering rules.

An important part of our approach is an "internal form" that is capable of representing the implementation at different levels of abstraction. Our system to support logic synthesis makes use of a graph-like internal data structure for storing the implementation as it progresses from the higher-level description to its final form, and transformations operate on this graph. All descriptions are synchronous models of hardware: there is an implicit clock and on each 'tick' the state of the machine is defined. The computation of the next state and outputs in terms of the current state and inputs proceeds between the ticks. There is a single organizational component: the 'box'. A box has input and output terminals which are connected by wires to other boxes. Each box also has a type, which may be primitive or may reference a definition in terms of other boxes. Thus a hierarchy of boxes can be used, and an instance of a high-level box such as a parity box can be treated as a single box or expanded into its next level implementation when that is desirable.

The Initial Implementation

The first step is to translate the functional specification into a network of high-level functions to form an initial, naive implementation. The difficulty of this task depends on the level of the functional specification. In our case local implementations of each specification language construct are substituted in a straightforward fashion. While the resulting initial implementation is in some cases extremely naive, the generator of this implementation is simple, which reduces the task of insuring that it produces correct implementations.

The Transformations

A library of transformations, which can be applied to the initial implementation or any implementation, is available. Not all transforms are applicable at every level, and there is some choice in selecting a particular application sequence. It is here that the interaction with the user is most important. Figure 2 summarizes the sequence of transforms used in this experiment; the application sequence might be very different in other experiments. Our hope is that after further experimentation we will find a set of transformations and an application sequence that produce acceptable implementations for a large class of chip specification with little or no user intervention.

We have made some decisions in ordering of the transforms. For example, there are three simplification steps shown in Figure 2. One might question whether simplification both before and after the generation of NANDs is necessary. The inclusion of both steps reflects a tradeoff between applying transforms as early as possible to reduce the size of the network to which later transforms are applied, and having fewer transforms. Another tradeoff exists in the writing of transforms themselves. For reasons of efficiency one might write one transform so that it called another after making a change which suggested that the other transform would apply. On the other hand, this makes the individual transforms more complicated and less flexible.

The synthesis process is an alternating sequence of descriptions and transformations. These intermediate descriptions and the transforms used to produce them are summarized below:

Level 1. The starting point is a naive implementation generated from the functional description in a straightforward fashion. In our example this is a network of 183 boxes (e.g. AND, OR, REGISTER, and PARITY). Box count is only one measure and is listed here merely to give the reader a feel for the effect of the various transforms.

Level 2. Here simple local transformations are applied to the initial implementation and substantially reduce the number of boxes. Examples of the transforms used are:

AND(a,NOT(a)) → 0
OR(a,NOT(a)) → 1
OR(a,AND(NOT(a),b) → OR(a,b)
XOR(PARITY(a_1,...,a_n),b) → PARITY(a_1,...,a_n,b)

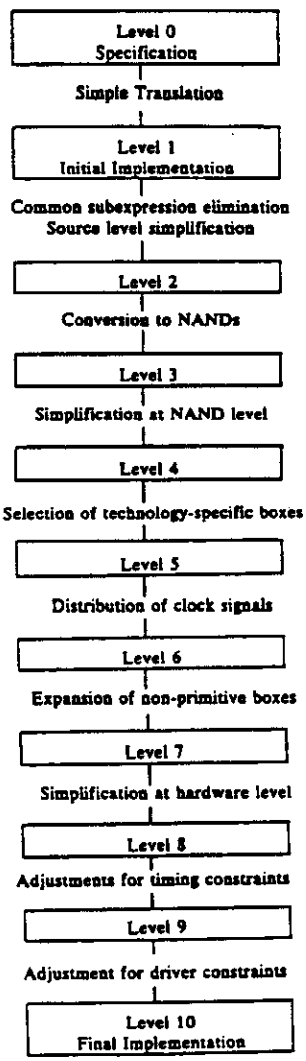


Figure 2 The steps of the synthesis experiment

These transformations, as well as the initial generation, may introduce constant inputs 0 and 1 for boxes, and may generate several copies of a box. Transformations are applied to eliminate these through constant propagation and common subexpression elimination, techniques borrowed from optimizing compilers. These changes, in turn, may produce boxes which have either no inputs or no outputs. These are successively removed in a technique analogous to unreachable code elimination. As a result of these steps the implementation is reduced to 83 boxes.

Level 3. The AND, OR, NOT, and most other operators of the initial description are replaced by NAND implementations. These transformations are local and may introduce unnecessary double NANDS that will be eliminated later. Also idealized senders and receivers are installed at the chip interface. Some higher-level operators such as EQ and PARITY are not replaced now but will be expanded later. At this point the implementation contains 174 boxes.

Level 4. Simplifying transforms are applied to each signal in the network. These transforms are local in that they replace a small subgraph of the network (usually five boxes or fewer) by another subgraph which is functionally equivalent but simpler according to some measure. They are used throughout the network until no more apply. Elimination of "double negation", a sequence of two single-input NANDs, is the transform most used at this step. This and another gate-reducing transform are shown in Figure 3.

When the nand transforms are applied globally the reduce the number of boxes to 158. However, a designer would notice that one signal network that is still overly complex. On close examination he would find that the fan-out of intermediate signals prevents the transforms from applying. He can intervene at this point, eliminate the fan-out by introducing duplicate logic, and reapply the transforms to further reduce box count and path length. Final box count is 153.

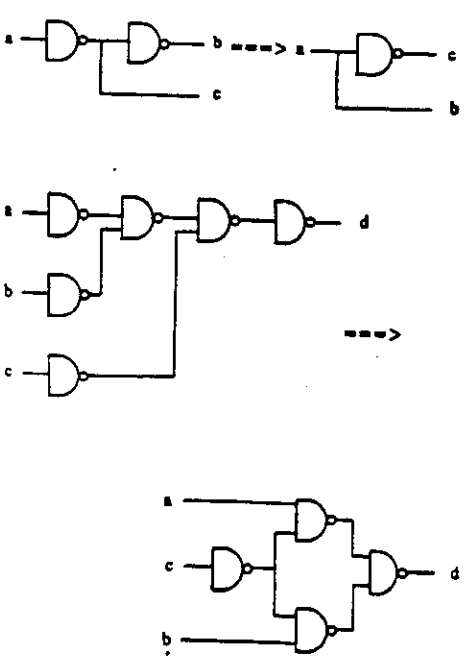


Figure 3 Examples of NAND Transformations

Level 5. At this level the first technology-specific transforms are applied. Specific networks of gates in the target technology are selected to implement the NANDs, idealized registers, drivers, and other boxes present in the previous level. Some manipulation may be necessary to match the fan-in of the actual primitives with that of the "idealized" boxes. Also the number of control and data lines of the idealized registers might exceed those actually available, and necessitate the generation of additional logic. The logic at this level is in terms of primitives used by the engineers in their implementations, but may not obey all timing, fan-out, and other technology restrictions. In some cases this is a many-to-one mapping, and the resulting size is 93 boxes.

Level 6. Up to this point the latches are assumed to be clocked by an implicit clock. Now that actual latches have replaced the idealized registers, actual clock signals must be attached. The technology file contains the rules for distributing the clock to the latches. The implementation now contains 134 boxes.

Level 7. At this point the higher-level operators such as PARITY are expanded in terms of the hardware primitives. The definitions of these boxes will have been synthesized themselves, either by hand or with automatic aid. For example, the two input EQ function can be replaced by a network of three NAND gates realized in the technology. After expansion there are 192 boxes.

Level 8. Replacement of boxes by lower level primitives may make further gate-saving simplifications possible, especially at the "borders" of the expansions and the previously existing logic. Technology-specific analogs of some of the transforms of Level 5 may be applied. Also, wired (dot) ANDs may be introduced if allowed by the technology; they may be used to eliminate gates if the inputs to a NAND-realized AND are not used elsewhere. The simplified implementation contains 159 boxes.

Level 9. The technology may specify a maximum and minimum number of gate delays allowed between latches. Though complete timing analysis may not be possible until after placement and wiring, some guidelines can be enforced by transforms at this level, by adding gates to short paths or by reducing numbers of levels to shorten long paths. Information about whether input signals change rapidly enough to make their delays critical, supplied with the chip interface information, may be used to avoid unnecessary delays. After this adjustment there are 188 boxes.

Level 10. The fan-out of each signal is adjusted to conform to the technology constraints, represented by inequalities in the technology file. Power requirements may require the source box of a signal with fan-out exceeding these limits to be duplicated to reduce this fan-out. In our experiment the box count remained at 188. While the 188 generated boxes are primitives for designers, they are in fact macros requiring several master-slice gates for implementation. The fully expanded implementation nearly fills the chip.

Results

The synthesized chip was compared with the existing implementation, and no significant differences were detected. In fact the synthesized implementation used five fewer gates than that of the engineer, and satisfied all technology constraints. An example of the differences was that two unnecessary delays included by the engineer were omitted in our synthesis. Another difference was in the use of dot ANDs. Considering that the experiment involved a single chip, that we had access to the manual implementation, and that we selected the order of application of transforms, it should not be a surprise that the two implementations were in such close agreement.

Future

During the coming year we plan to continue the development of the prototype synthesis system and to use it in conducting further experiments. We plan to look at more ambitious chips -- chips that have required minimization or that have caused long path problems, when implemented manually. We may find that additional user direction will be necessary. This may require additional measures and transforms, but our hope is that new chips will not always require additional transforms and that we can begin to develop sequences of transforms and higher-level transforms that will make the total task more automatic. In addition, we will explore:

- multi-chip synthesis -- starting with a functional specification that requires several chips, developing additional measures and transforms that will trade resources across chip boundaries.

- alternative technologies -- looking at the sensitivity of synthesis to alternative technologies and determining how the synthesis system could respond to changing technology.

- engineering changes -- examining how such a synthesis system could respond to EC's where minimum, local changes are highly desirable.

- transform specification -- looking at how transforms could be described at a high level and compiled for efficient application.

- transform correctness -- considering what properties should be proved of transforms such as function preserving or power decreasing, and demonstrating how such proofs can be accomplished.

Summary

We are in the midst of taking what we believe is a new approach to the old problem of logic synthesis and are encouraged by our preliminary experiments. We have built a prototype synthesis system that allow us to perform more ambitious experiments to validate our approach. Our hope is that computationally manageable techniques based on local transformations can be applied to improve naive implementations to acceptable ones. This could greatly shorten processor development and validation times.

Acknowledgements

We would like to thank William van Loo for many helpful discussions on master-slice chip design. Also Leonard Berman, Louise Trevillian, and Thomas Wanuga have made valuable contributions to the design and implementation of the prototype synthesis system.

References

[Bar73] Barbacci M, "Automated Exploration of the Design Space for Register Transfer Systems", PhD Thesis, CS Dept., Carnegie-Mellon University, 1973.

[Bre72] Breuer M A (Editor), *Design Automation of Digital Systems*, Prentice-Hall, 1972.

[Chu65] Chu Y, "An ALGOL-like Computer Design Language", *Communications ACM*, Oct 1965.

[Dar69] Darringer J A, "The Description, Simulation, and Automatic Implementation of Digital Computer Processors", Ph.D. Thesis Carnegie-Mellon University 1969.

[Die71] Dietmeyer D L, *Logic Design of Digital Systems*, Allyn and Bacon, 1971,1978.

[Dul68] Duley J R, "DDL -- A Digital Design Language", Ph.D. Thesis, University of Wisconsin 1968.

[Dul69] Duley J R and D L Dietmeyer, "Translation of a DDL Digital System Specification to Boolean Equations", *IEEE TC Vol C-18*, April 1969.

[Fri68] Friedman T D and S C Yang, "Methods used in an Automatic Logic Design Generator(ALERT)," *IEEE TC Vol C-18*, July 1968.

[Fri70] Friedman T D and S C Yang, "Quality of Design From an Automatic Logic Logic Generator(ALERT)," Proc. 1970 Design Automation Conference.

[Haf78] Hafer L J and A C Parker, "Register-Transfer Level Digital Design Automation: The Allocation Process", Proc. 15th Design Automation Conf., 1978.

[Hil73] Hill F J and G R Peterson, *Digital Systems: Hardware Organization and Control*, Wiley, 1973.

[Mes68] Mesztenyi C K, "Computer Design Language Simulation and Boolean Translation", TR68-72, CS Dept., Univ. of Maryland, 1968.

[Nak78] Nakamura S et.al., "LORES - Logic Reorganization System", Proc. 15th Design Automation Conf., 1978.

[Sch62] Schorr H, "Toward the Automatic Analysis and Synthesis of Digital Systems" Ph.D. Thesis, Princeton, 1962.

[Sie76] Siewiorek D P and M R Barbacci, "The CMU RT-CAD System -- An Innovative Approach to Computer Aided Design", Proc. AFIPS Conf. Vol. 45, 1976.

[Sno78] Snow E A, "Automation of Module Set Independent Register-Transfer Level Design", PhD Thesis, EE Dept., Carnegie-Mellon University, 1978.

[Tho77] Thomas D E, "The Design and Analysis of an Automated Design Style Selector", PhD Thesis, EE Dept., Carnegie-Mellon University, 1977.

PTO - 948 (Rev. 8-82)

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE

ATTACHMENT TO PAPER NUMBER 3

GROUP 234

S.N. 143821

NOTICE OF PATENT DRAWINGS OBJECTION

Drawing Corrections and/or new drawings may only be submitted in the manner set forth in the attached letter, "Information on How to Effect Drawing Changes" PTO-1474.

A. ☑ The drawings, filed on 1-13-88, are objected to as informal for reason(s) checked below:

1. ☐ Lines Pale.
2. ☐ Paper Poor.
3. ☑ Numerals Poor. (FIG 1, 3, 7, 8, 11, 12, new 15)
4. ☐ Lines Rough and Blurred.
5. ☐ Shade Lines Required.
6. ☐ Figures Must be Numbered.
7. ☑ Heading Space Required. 2" at TOP
8. ☐ Figures Must Not be Connected.
9. ☑ Criss-Cross Hatching Objectionable. FIG 1C
10. ☐ Double-Line Hatching Objectionable.
11. ☐ Parts in Section Must Be Hatched.
12. ☐ Solid Black Objectionable.
13. ☐ Figure Legends Placed Incorrectly.
14. ☐ Mounted Photographs.
15. ☐ Extraneous Matter Objectionable. [37 CFR 1.84 (1)]
16. ☐ Paper Undersized; either 8½" x 14", or 21.0 cm. x 29.7 cm. required.
17. ☐ Proper A4 Margins Required:
    ☐ TOP 2.5 cm. ☐ RIGHT 1.5 cm.
    ☐ LEFT 2.5 cm. ☐ BOTTOM 1.0 cm.
18. ☑ Other: ZIP-A-TONE NOT ACCEPTABLE
    ... FIG 1C FIGURE LEGENDS TOO SMALL (FIG 10)
    ... JAGGED LINES ARE NOT ACCEPTABLE (FIG 4, 5,

B. ☑ The drawings, submitted on 1-13-88, are so informal they cannot be corrected. New drawings are required. Submission of the new drawings MUST be made in accordance with the attached letter.

RCL000184

#4

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Hideaki Kobayashi and Masahiro Shindo
Serial No. 143,821
Filed: January 13, 1988
For: KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS

Group Art Unit: ~~236~~ 234

The Honorable Commissioner of Patents and Trademarks
Washington, D.C. 20231

January 17, 1989

POWER TO INSPECT

Sir:

Please permit Robert L. Smith, or his representatives, to inspect the above-entitled application, and to make copies of any of the papers that they may desire.

Respectfully submitted,

Raymond O. Linker, Jr.
Registration No. 26,419

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, North Carolina 28234
(704) 377-1561
ROLjr:klc
Our File 3868-2

89 JAN 18 PM 5:35

#5

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

H. Kobayashi & M. Shindo
Serial No. 143,821
Filed: January 13, 1988
For: *Knowledge Based Method and Apparatus For Designing Integrated Circuits Using Functional Specifications*

Group Art Unit: 234

Examiner V. Trans

April 18, 1989

The Honorable Commissioner of
Patents and Trademarks
Washington, DC 20231

RECEIVED
APR 21 1989
GROUP 230

CITATION UNDER 37 CFR 1.97

Sir:

Attached is a form PTO-1449 listing several documents which may be considered material to the examination of the above identified application. A copy of each cited document is also enclosed. It is requested that these documents be considered by the Examiner and officially made of record in accordance with the provisions of 37 CFR 1.97 and Section 609 of the MPEP.

Respectfully submitted,

Raymond O. Linker, Jr.
Registration No. 26,419

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, North Carolina 28234
Telephone: (704) 377-1561
Our File No. 3868-2
ROLjr:lma

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner of Patents and Trademarks, Washington, D.C. 20231, on April 18, 1989

Raymond O. Linker, Jr.
Reg. No. 26,419

April 18, 1989
Date of Signature

Sheet 1 of 1

Form PTO-1449 U.S. Department of Commerce Patent and Trademark Office	Atty. Docket No. 3868-2	Serial No. 143,821
LIST OF DOCUMENTS CITED BY APPLICANT (Use several sheets if necessary)	Applicant Hideaki Kobayashi & Masahiro Shindo	
	Filing Date January 13, 1988	Group 234

MAIL ROOM APR 20 1989 PAT. & TRADEMARK OFF.

U. S. PATENT DOCUMENTS

Examiner Initial		Document Number	Date	Name	Class	Subclass	Filing Date if Appropriate
	AA						
	AB						
	AC						
	AD						
	AE						
	AF						
	AG						
	AH						
	AI						
	AJ						
	AK						

FOREIGN PATENT DOCUMENTS

		Document Number	Date	Country	Class	Subclass	Translation Yes	Translation No
	AL							
	AM							
	AN							
	AO							
	AP							

OTHER DOCUMENTS (Including Author, Title, Date, Pertinent Pages, Etc.)

V1	AR	✓Trevillyan - Trickey, H., *Flamel: A High Level Hardware Compiler*, IEEE Transactions On Computer Aided Design, March 1987, pp. 259-269 14
V1	AS	Parker et al., *The CMU Design Automation System - An Example Of Automated Data Path Design*, Proceedings Of The 16th Design Automation Conference, Las Vegas, Nevada, 1979, pp. 73-80 14
	AT	RCL000187

EXAMINER	DATE CONSIDERED
V. TRANS	6/30/89

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN, VOL. CAD-6, NO. 2, MARCH 1987 RX #2

Flamel: A High-Level Hardware Compiler

HOWARD TRICKEY, MEMBER, IEEE

Abstract—This paper describes the design and implementation of a high-level hardware compiler called *Flamel*. Ordinary Pascal programs are used to define the behavior required of the hardware. Flamel undertakes to find parallelism in the program, so it can produce a fast-running implementation that meets a user-specified cost bound.

A number of program transformations create sections of code with more parallel computations than the original program has. A novel feature of Flamel is a method for organizing the search for the transformations that best satisfy the goal. Another new algorithm is one for "expression height reduction": rewriting an ensemble of expressions using algebraic properties in order to compute the expressions faster.

An implementation of Flamel has been completed. The output is a description of a datapath and a controller, and at a sufficient level of detail so that good area and execution time figures can be estimated. On a series of tests, Flamel produces implementations of programs that would run 22 to 200 times faster than an MC68000 running the same programs, if the clock cycles were the same. The tests also show that a wide range of time-area tradeoffs are produced by varying the area constraint.

I. Introduction

MUCH OF THE early work in hardware synthesis took hardware descriptions that were mostly *structural*, meaning that the input described the architecture of the required system and the synthesis systems instantiated that architecture in some technology. More recently, it has become feasible to build compilers that accept *behavioral* descriptions, where the input simply describes what the required system needs to do. Such compilers need to choose an architecture; after that, the techniques developed for the structural synthesizers can be used. This paper describes a behavioral hardware compiler called Flamel.[1]

A typical way of describing behavior is to write a program in an ordinary computer language, with some convention as to how to communicate with the "outside world." Roughly speaking, the required system needs to act the same way across the communication channel that the program does. The hardware compiler can rearrange the program and have calculations done in parallel, so long as the proper values are written to the outside world at the proper times. Some examples of compilers that operate this way are: the CMU-DA project [1], particularly the Design Automation Assistant [2], [3] portion; Arsenic [4]; the USC Design Automation project [5]; the AT&T Bell Labs VLSI Design Automation Assistant [6]; and SC [7]. There are some similarities between Flamel and these other systems. The thing that distinguishes Flamel from the others is extensive modification of the input program, together with an algorithm for deciding which modifications yield the fastest-executing chips while meeting a user-supplied cost constraint.

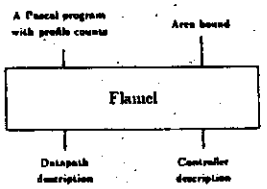


Fig. 1. Flamel's operation.

An overall picture of Flamel's operation is shown in Fig. 1. The user provides a Pascal program together with execution frequency counts for a typical execution of the input program. The other user input is a number saying roughly how much hardware is allowed. The output is a design for hardware that will perform the same function as the Pascal program.

II. The Compilation Problem

The problem that Flamel attempts to solve is:

> Given a program *P*, find a hardware implementation with the same external behavior as *P*. The implementation should minimize the execution time of the implementation running on typical data, while meeting a user-supplied constraint on the cost of the hardware implementation.

This section will define *hardware implementation* and *external behavior*, and explain how Flamel estimates execution time and hardware cost.

A. Hardware Model

The general model for a circuit produced by Flamel is that of a synchronous digital machine consisting of a *datapath* and a *controller*. The datapath consists of *functional units* (ALU's, adders, registers, I/O pads, etc.) interconnected by busses. Functional units source, sink, hold, and operate on data words, where a word contains some number of bits. A given bus may receive data from and give data to several functional units. This means that

Manuscript received February 16, 1986; revised November 12, 1986. This work was done while the author was at Stanford University. Supported by DARPA contract N00039-83-C-0136.

The author is with AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974.

IEEE Log Number 8612997.

[1]After a medieval alchemist said by some to have achieved the "Great Work" of gold-making.

0278-0070/87/0300-0259$01.00 © 1987 IEEE

RCL000188

multiplexers may be needed on functional unit inputs and outputs in order to regulate the bus traffic. Note that this model allows anything on the spectrum between global busses and dedicated connections. Collectively, functional units and busses are called *resources*.

The controller is a finite-state machine. The inputs to the machine come from the datapath's functional units—information like the low-order bit of a register or the status bits of an ALU. The outputs control the datapath, setting the function for multifunction units, and determining multiplexer selector settings.

It is useful to impose a structure on the finite-state machine. Flamel builds a machine consisting of *blocks*, where each block is what would be called "straight-line code" in an ordinary compiler. (Sometimes the term "block" is used in hardware literature to denote a subcircuit, but we don't use the term in that sense. A Flamel block is not usually implemented as a distinct circuit, but rather, as some portion of microcode memory.) There are a fixed number of *cycles* in a block, and a block always executes each cycle in turn so that it finishes some fixed, data-independent time after it is entered. Some of the operations in a cycle might be done only conditionally, so it is not quite like "straight-line code" in that sense. When a block finishes, it passes control to another block (*activates* it), perhaps using a datapath output to choose among several possibilities.

Given a hardware implementation for a program P, Flamel tries to optimize the *estimated running time of* P, denoted $T(P)$. For a block B, let $freq(B)$ be a *frequency count*: an estimate of how many times the block executes in a typical run of the input program. That number is derived from data that the user must supply (described below). Also, let $H(B)$ be the number of cycles required to execute block B. Then

$$T(B) \stackrel{def}{=} freq(B) \times H(B)$$

and $T(P)$, the time for a whole program, is

$$T(P) \stackrel{def}{=} \sum_{B \in bls(P)} T(B)$$

where $bls(P)$ is the set of blocks implementing P. Typically, the user runs the (software) program on sample input data, and uses a profiling tool to find out how often each statement executes. If those numbers are used to derive the $freq(B)$ values, then $T(P)$ will be the number of cycles that the Flamel-generated chip will need to execute when presented with the same input data.

The cost of a hardware implementation for a program P, denoted $C(P)$, depends on the implementation technology. The prototype Flamel is targeted to a VLSI implementation, and $C(P)$ is an estimate of the chip area. We adopt a layout scheme that has found some success in other projects: both MacPitts [8] and Shrobe's datapath generator [9] used a bit slice architecture. The functional units are arranged in a vertical line and the busses run in between each bit of the units. The busses needn't run the full length of the line—only as far as necessary. Thus, you get the effect of several local busses sharing the same routing track.

Flamel's current output describes a data path by naming the functional units in the order they appear, and the busses with their attachments. Flamel has a table with the height of each possible functional unit (assuming predesigned cells), and other data such as the width of all the cells, and the height and width increments incurred by additional wiring tracks. From that, it is possible to make a very good estimate of the area of a bit slice. The output also describes a controller, by giving the states and transitions of a finite-state machine. There are other programs that generate VLSI layouts for the controllers (SLIM [10], and a *regular expression compiler* [11]), but the prototype Flamel doesn't get feedback on how big the controllers might be, so the controller area isn't included in its cost estimate.

B. Pascal as a Behavior Specification Language

The input to Flamel is a Pascal program. Suppose that a statement like "**read**(x)" is executed at some point during the running of the input program. Then the hardware implementation should have an input port with enough pads to read all the bits of x; a value for x will be read at some point during the running of the chip. Similarly, a statement like "**write**(x)" implies an output port in the hardware implementation.

A hardware implementation of a Pascal program will be *valid* if the following is true:

> Given any set of input data, the program and the hardware implementation must do I/O in the same order, and with the same values.

It is convenient to allow more than one value to be read or written in a single cycle. Flamel uses this convention: a statement like "**read**(x, y)" means that the hardware implementation should read x and y at the same time. Thus, there need to be two input ports in such a case. For reads that aren't to occur simultaneously, only one port need be provided (in analogy to one input stream for a software program). A similar convention holds for writes.

The prototype Flamel imposes some restrictions on the input programs.

- The program must be a single, parameterless, nonrecursive procedure.
- The only datatypes allowed are: integer, boolean, char, and subranges and one-dimensional arrays of these.
- Multiplication, division, and mod are not allowed, except by constants that are powers of 2.

Only the first restriction requires much of an extension to the techniques developed for Flamel.

Flamel converts the input program into an internal form, dubbed *dacon* (the dataflow/control-flow). Some proper-

ties of dacons need to be explained before Flamel's algorithms can be understood.

The dacon representation of a program is a combination of two data structures commonly used in software compilers [12]. The program is divided up into *basic blocks*: sections of code with no jumps into or out of them except at the beginning and end. There is a *block graph* (often called a *flow graph*), whose nodes represent blocks and whose arcs represent the "transfers-control-to" relation. For each block, there is a *directed acyclic graph* (DAG) that gives the detailed calculations of the block. The nodes of the DAG are operators (*add*, *constant source*, *pad read*, *array element read*, etc.) and the edges carry values.

The reason for the name "dacon" is that the representation implies a hardware implementation where the blocks operate in a dataflow-like manner (i.e., calculations can be done as soon as input values are ready), but control passes between blocks as in ordinary programs.

One important feature of the dacon representation of a program is that the local nonarray variables of a procedure are not assumed to reside in fixed places. Instead, all the needed variables are passed from block to block. Some of the DAG leaves are *input shadow* nodes, representing the places where needed values are to be found. Some of the DAG roots are *output shadow* nodes, representing places where the values needed by the following block(s) are to be put. A block only has to pass a variable's value on if that value is *live* (potentially used by a direct or indirect successor block). Treating variables this way provides automatic register allocation, when combined with Flamel's resource allocation mechanism.

A complication with DAG's is the need to deal with execution order constraints. For instance, if the original source program has one read statement and then another, we are supposed to ensure that these happen in the hardware implementation. Similarly, a write to an array element $a[i]$ and either a read or write of $a[j]$ must occur in the same order as in the source program, unless we know that the subscripts can't be equal. The method for enforcing such constraints is to use special DAG edges called *control edges*. A control edge from node x to node y means that node x must execute before node y in any hardware implementation of the block.

There are many similarities between the dacon data structure and the "Value Trace" used in the CMU-DA system [13]. Dacons have a better separation between control-flow and dataflow, and that seems to simplify the global optimization process. But the Value Trace is close enough that it could probably have been used instead.

III. Choosing the Block Structure

The initial dacon for a program is built by dividing the program into basic blocks—sections of code that are always entered at the beginning and exit only at the end. Now if you look at the calculations going on inside such a block, you find that sometimes there are operations that can be done simultaneously because neither depends on the other, directly or indirectly, for inputs. Such operations constitute potential parallelism.

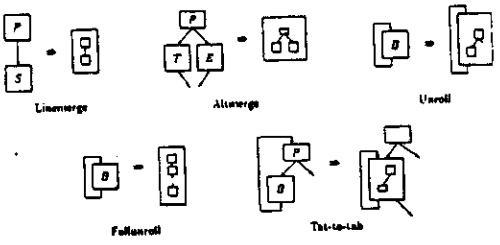


Fig. 2. Flamel's block-level transforms.

People have done empirical studies of the average amount of potential parallelism in a basic block of a typical high-level language program. It turns out that there isn't very much: typically, the studies show that the average "degree of parallelism" (number of operations done at any given time) is at most two or three [14]. Thus, a compiler that exploits only basic block parallelism isn't going to be able to produce a wide range of time/area tradeoff implementations.

Flamel's way out of this problem is to transform the dacon to make bigger blocks with more calculations in them. This turns out to yield impressive amounts of parallelism—usually a parallelism degree of five to ten.

A. Flamel's Block-Level Transforms

The five block-level transforms in Flamel's repertoire are shown in Fig. 2. Each creates a single block whose DAG is formed in a simple way from the DAG's of the original block(s).

Linemerge: Connect the outputs of P to the inputs of S.

Altmerge (Alternative merge): Attach both T and E to the outputs of P, and then use multiplexers controlled by P's exit condition to choose the correct outputs to pass on to the common successor of T and E. If T or E have nodes with side effects (e.g., writing an array), those nodes need to be flagged so that they execute only when they should. There is also a variant where T or E is missing.

Unroll: Attach a copy of B to the outputs that lead to restarting the loop, and use multiplexers to choose the correct outputs (from the first or second body) to pass on to the loop's successor.

Fullunroll: Like unroll, but use as many copies as there are loop iterations (when that number is known at compile time).

Tat-to-tab: A "test-at-top" loop has two blocks, the top one testing to see if the bottom—the body—should execute. The unroll transforms only work on the one-block "test-at-bottom" loop, so this transform creates one by incorporating the test block at the end of the body block.

It is possible to define a formal notion of *computational equivalence* of dacons, and show that the above transformations preserve it [15].

As an example, consider the *mmult* procedure, which calculates ab mod n, given a, b, and n. Fig. 3 shows the code and the initial block graph; the boxes surrounding sections of code show approximately what happens in each block. The first transformation on Fig. 3 might be an altmerge of blocks 3 and 4. That would produce a block that calculates $s + a$ and $s + a - n$ and passes one or the other along, depending on the concurrently calculated $s + a \geq n$. Then the newly-formed block could be linemerged with block 5. Further transformations could eventually produce one single block for the whole program.

B. The Transform Tree

The transformations introduced in the previous subsection are not particularly new. What *is* new in Flamel is a controlled way for deciding which transforms to apply. Central to the method is a data structure called a *transform tree*. The *transform tree* has the program's original basic blocks as leaves, and the internal nodes represent blocks that were formed during the transformation process. They are labeled with letters to say how they were formed from their children. For example, Fig. 4 shows the transform tree for *mmult*. (For the moment, ignore the numbers surrounding the blocks.)

The significance of the transform tree is that it succinctly represents the search space Flamel explores when it tries to decide on the block graph of the dacon it will implement. Conceptually, Flamel chooses the blocks to implement as follows:

> Either choose the root block to implement, or choose implementations for its children (recursively) and connect them as dictated by the transformation-type label.

Choosing the implementation this way is very attractive, for two reasons. First, the total number of blocks examined is small—about twice the number of original blocks. And second, it avoids "local optimum" problems, where one stops transforming because any single extra transform worsens things, even though a worsening transform might be necessary to reach a global optimum.

For the most part, there is only one transform that can be done involving any given block. There are four exceptions to this, shown in Fig. 5. Flamel chooses the linemerge 1–2 in case (a), the altmerge in cases (b) and (c), and several unrolls before a fullunroll in case (d). The important case is (b); the choice there is based on the experience that a linemerge doesn't often increase the amount of parallelism, since the critical computation path in the predecessor block often connects to the critical path in the successor. At any rate, the linemerge can be done after the altmerge, resulting in the same blocks as if the transforms were done in the other order, so the disambiguation choice doesn't have a huge effect on the final transform tree.

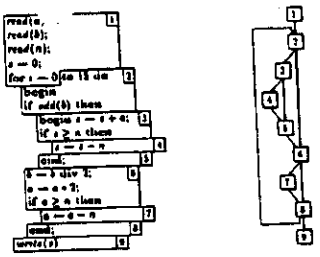


Fig. 3. *mmult* and block graph.

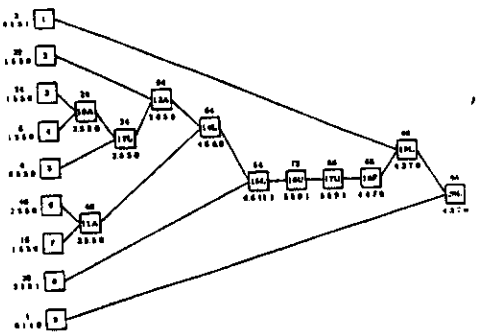


Fig. 4. Transform tree for *mmult*.

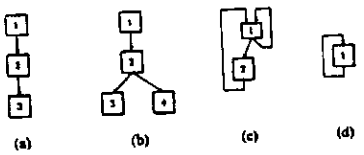


Fig. 5. Ambiguous cases for transforms.

A problem is that, early in the transformation process, the graph may not look like Fig. 5(b), yet it may after some portion of the graph collapses into one block. To avoid doing the linemerge too early, Flamel only does linemerges when none of the other transformations apply to the second block. Flamel builds the transform tree assuming that fullunrolls can always be done, postponing the actual building of the DAG's for the transformed blocks. This tactic means that any dacon derived from a Pascal procedure that doesn't use **goto's** or **case** statements will be reduced to a single block, as evidenced by the relation between the Pascal compound statement forms and the transform patterns. Procedures that have **goto's** can still be handled by Flamel, but there may be less opportunity to find parallelism.

C. Choosing the Blocks

After the transform tree has been made, the next step is to pick the set of nodes that will be used in the dacon to

be implemented. Recall that the goal is to find an approximate solution to the *mintime*(P, c) problem—that of minimizing execution time while meeting cost constraint c. The *mintime*(P, c) problem seems to be extremely difficult to handle, because each block will have its own cost-time tradeoffs. Flamel's method is to solve a simpler problem, called *rmintime*. Instead of meeting a cost bound, the implementation must meet a *resource* bound:

$$rmintime(P, R) \stackrel{\text{def}}{=} \min T(P), \qquad reses(P) \subseteq R$$

where R is a set of resources, *reses*(P) is the set of resources used in the implementation of P, and the minimization is over all possible implementations. So a typical *rmintime* problem might be: find a minimum time implementation of program P that uses three ALU's, five registers, eight constants, and two I/O ports.

The great thing about the *rmintime* problem as opposed to the *mintime* one is that each block can be chosen independently. Since the blocks don't overlap execution,[2] resources can be reused from block to block, so that if each block meets the resource bound independently, then the whole program will.

Now Flamel's block-choosing method can be presented.

1. Use the cost bound c to derive a resource bound R. Flamel's current method is to allocate approximately equal portions of c to each of the resource classes: ALU, register, constant, bus. A typical cost for each of these resources is estimated and divided into the cost portion to get the number of each in R. The number of I/O pads and register files allowed is dictated by the program (we have no choice).

2. Build DAG's for the blocks represented by nodes in the transform tree. Use a method to be presented later to solve *rmintime*(B, R) for each block B and yield cost and time estimates for each block.

The DAG building proceeds bottom-up, using a postorder traversal of the transform tree. Don't bother building a DAG if the children blocks exceed the resource bound R, since such a DAG will almost certainly also exceed the resource bound. Also, a block may be unimplementable because it is a fullunroll of a loop that cannot be fully unrolled.

3. For each block B that meets the resource bound, calculate

$$besttime(B) = \begin{cases} T(B) & \text{if } B \text{ is a leaf} \\ \min\left(T(B), \sum_{C \text{ child of } B} besttime(C)\right) & \text{otherwise.} \end{cases}$$

This number represents the smallest total amount of time that will be spent in block B if the fastest implementation is chosen. The "fastest implementation" uses either the DAG for B, or the fastest implementation (defined recursively) of its children. Let $T(B) = \infty$ if B doesn't meet the resource bound.

4. Finally, reconstruct the dacon that yields the fastest implementation. Do this with a preorder traversal of the dacon that was left after building the transform tree (probably a single block). When visiting a node B, see if *besttime*$(B) = T(B)$; if so, B should be part of the chosen dacon, so return without visiting the children. Otherwise, *undo* the transform that led to B: remove B from the current block graph and replace it with its children, interconnected as dictated by the transform type.

Fig. 4 gave the transform tree for *mmult*, derived by using the rules just given. If Flamel is given a large resource bound, then all of the blocks have DAG's built for them, and Fig. 4 shows the $T(B)$ value[3] (on top) and estimated resource set (on bottom) for each node. The resource set is given by a string of the form $w.x.y.z$, where w, x, y, and z are the numbers of ALU's, registers, busses, and constants, respectively. For example, block 3 has a DAG that can be executed in three cycles, and thus $T(B_3) = freq(B_3) \times H(B_3) = 8 \times 3 = 24$; and, Flamel estimates that one ALU, five registers, and five busses suffice to implement it.

Many people have proposed program transformation systems without a global strategy for deciding which transforms to apply. Such systems would probably get stuck in locally optimal configurations. In the *mmult* example, the 2–12L altmerge yields a slower block, but doing it enables the other transformations, yielding a final implementation consisting of the single block 20L. The final implementation chosen by Flamel will execute in 66 cycles versus 140 for the locally optimal implementation (blocks 1, 2, 12L, 11A, 8, 9).

The user can supply a cost bound, nominally the height of the bit slice, and Flamel derives resource bounds from that. A bound allowing only two ALU-class functional units will cause Flamel to omit trying to implement blocks 15–20, and the best-time block set satisfying the bounds turns out to be the locally optimal one mentioned earlier. Note that the cost bound only loosely defines the architecture. Flamel's resource allocation procedure chooses two adders, two shifters, and one ander for this example—it turns out to be cheaper than two ALU's.

IV. DAG Transforms

Flamel needs as a subroutine a method for solving the *rmintime* problem for blocks. Recall that the *rmintime* problem is to minimize the $T(P)$, the expected amount of time spent executing a program. For a block B, $T(B)$ is the product of the estimated block frequency and the number of cycles it takes to execute the block, $H(B)$. When working on a single block, we can equally well say that the *rmintime* problem is the minimize $H(B)$.

[2] At least, not in the prototype Flamel.

[3] The frequency values assume that the If tests are true half of the time.

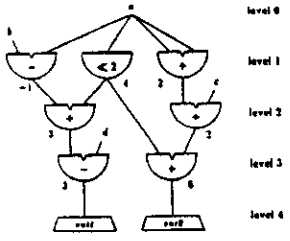


Fig. 6. DAG for level compression.

More precisely, the problem is to take a DAG and decide in which cycle each node is to start performing its computation. The *schedule* needs to satisfy certain properties: a node must be scheduled no earlier than some specified delay after each of its predecessors, and the resource constraints have to be respected in each cycle. A further aspect to the problem is that certain transformations may be applied to the DAG, transformations allowed by the algebraic properties of the operators involved. This process is called *height reduction*, since the number of cycles used is the DAG height when all operators use one cycle to complete.

A. Height Reduction

The problem of reducing DAG height has been much studied in the special case where the DAG is a tree and the operator nodes are $+$, $-$, $\times$, and $/$. See Chapter 2 of Kuck's book [16] for a survey. Unfortunately, the restriction to trees makes those algorithms essentially useless for Flamel. Height reduction is only effective on large graphs, and the only large graphs created by Flamel are those created by loop unrolling. The DAG's of unrolled loops can be turned into trees by copying shared subexpressions, but they usually grow exponentially if that is done. Another problem with the published literature on height reduction is that it doesn't say how to deal with multiplexers.

Flamel incorporates a new height reduction algorithm, called *level compression*. The idea is to repeatedly pick some node in the DAG and shorten the distance between that node and the roots (sinks). Figs. 6 and 7 show how a node is *moved later*.[4] The numbers near the nodes in Fig. 6 are *contribution factors with respect to a*: informally, if an algebraic expression were written for the value calculated by a node g, the variable a would appear multiplied by the contribution factor of g. (Actually, contribution factors are defined inductively by saying that a has a contribution factor of 1, and giving rules of propagating contribution factors through nodes [15].) The move is accomplished by replacing a with 0 and adding compensating multiples of a according to the contribution factors at some level as late as possible. Fig. 7 shows the result; there is no height reduction here, but there would be in a larger example, such as a block arising from loop unrolling. Flamel performs *constant folding*—simplifications due to some or all inputs to a node being constant—yielding a much simpler DAG from Fig. 7.

[4]The nodes labelled "$\ll n$" mean "left shift n bits."

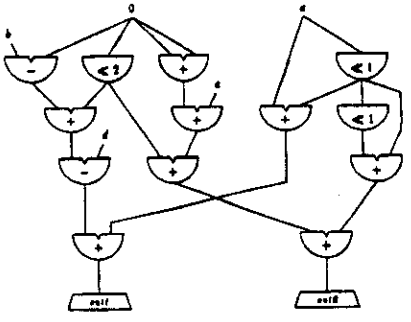


Fig. 7. DAG after level compression.

The method for choosing the node to move is basically: look for a node about halfway along a critical path (a path where all the time constraints are tight). As height reduction proceeds, certain nodes are *frozen* to prevent them from taking part in further reductions: nodes created for contribution calculations, nodes that contributions are added to, and nodes on paths from frozen nodes to sinks. Then nodes chosen for later moving must be about halfway along nonfrozen portions of critical paths. Without this freezing mechanism, the height reduction procedure usually cannot do better than approximately halve the original height. The moving process stops when there is no further height reduction or when the resource bounds are exceeded at some level. Care has to be taken to avoid doing transformations that increase the DAG height. For details, see Trickey [15].

One of the central methods of many parallelizing compilers is to examine loops to see if they have a special form, and then use one of a library of transformations to convert the loop into a more parallel form. For example, a *parallel prefix sum* calculation like

$$\text{for } i \leftarrow 2 \text{ to } n \text{ do } a[i] \leftarrow a[i-1] + b[i]$$

has the form of a *linear recurrence*, and a technique such as Kogge and Stone's [17] can calculate all of the $a[i]$ in $O(\log n)$ time. If the loop contains an IF statement, the method of Banerjee *et al.* [18] might be used. Flamel takes a different approach: rather than looking for special cases, all loop parallelization is done by loop unrolling followed by DAG height reduction. This approach yields the fastest possible calculation of the above loop, and it has the advantage that it would also work on a partially unrolled loop (for less parallelization when the cost bound dictates). It would also work if there were other calculations in the loop that were not in linear recurrence form, if those other calculations were not the bottleneck. Fig. 8 shows

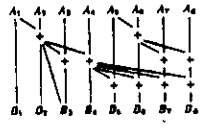


Fig. 8. Level compression height reduction on parallel prefix sum.

the result of level compression height reduction of the fully unrolled parallel prefix sum loop.

V. Hardware and Control Generation

After Flamel has chosen the dacon to implement, it needs to tackle the job of actually scheduling the nodes in all of the DAG's and assigning hardware resources to implement the operations and value transfers. Registers are resources, so register allocation occurs here too. In the prototype Flamel, only registers can hold values for more than one cycle, so registers are sometimes needed to hold intermediate values.

Flamel uses a method called *folding*, similar to a method used by Hitchcock and Thomas [19]. The basic idea is to start out with the *earliest possible* schedule and a resource assignment that gives every node its own functional unit and every edge its own bus. Then, pairs of resources that perform the same function, or could be generalized to perform the same function, are *folded* together into one resource if they aren't being used simultaneously. When there is a choice of pairs to fold, the pair yielding the best expected improvement in the hardware cost is chosen.

Flamel goes a step further than Hitchcock and Thomas. If there are no more possible folds, yet the estimated hardware cost is still more than the user allows, then Flamel *lengthens* the schedule: it chooses a pair of resources that could be folded if certain DAG operations didn't overlap in time, and then it forces those operations into different cycles by adding judicious control edges and rescheduling.

For an example of this resource allocation procedure consider the *mag* program shown in Fig. 9. This is a program for approximating $\sqrt{a^2 + b^2}$, translated from the MacPitts input given by Southard [20]. A comment follows the first statement of each basic block, giving the "frequency" fed to Flamel for that block. With no resource limit, Flamel chooses a single-block implementation for *mag*, with the DAG shown in Fig. 10. The rectangles with round ends are registers, added because none of the functional units have output buffers. Some nodes have multiplexers on their inputs (marked by lines across the top): the left or right input is used depending on whether the value shown feeding the end is 1 or 0, respectively. Each level of Fig. 10 will be a different cycle during execution. Notice how the two separate read statements have forced the pad reads (*in* nodes) to be on different levels.

Given the DAG of Fig. 10, the resource allocation procedure comes up with the hardware resources shown in Fig. 11: input pad, three registers, negater, adder/subtractor/comparitor, constant source, barrel shifter, output pad, and four busses. Again, input multiplexers are shown as lines across the top, with choices controlled via control lines (not shown). As an example of folding, the *aab*, *bab*, *g*, and *sqs* registers of Fig. 10 have all been folded into the *other* register of Fig. 11, but Flamel's cost tables said it was cheaper to leave *a* and *b* in separate registers (connections to several busses are almost as expensive as registers). The DAG edges out of the pads, *aab*, *b*, and *g* were folded into the leftmost bus of Fig. 11 and the edges out of *bab*, *sqs*, and '−' were folded into the rightmost bus. Those bus connections helped determine that it was free to use the *other* register for *sqs*.

The final step is to generate description files for the datapath and controller. Flamel uses an adaptation of Kernighan and Lin's heuristic procedure for partitioning graphs to choose the top-to-bottom ordering of the functional units in a bitslice [21]. The goal is to allow the

```
procedure mag;
  var a, b, aab, bab, g, t, sqs, res: integer;
  begin   {1}
  read(a);
  read(b);
  if (a < 0) then aab := -a   {.5}
  else aab := a;   {.5}
  if (b < 0) then   {1}
    bab := -b   {.5}
  else bab := b;   {.5}
  if (aab > bab) then   {1}
    begin   {.5}
    g := aab; t := bab
    end
  else begin   {.5}
    t := aab; g := bab
    end;
  sqs := (g - g div 8) + t div 2;   {1}
  if (g > sqs) then res := g   {.5}
  else res := sqs;   {.5}
  write(res)   {1}
  end;
```

Fig. 9. Flamel input for magnitude approximation.

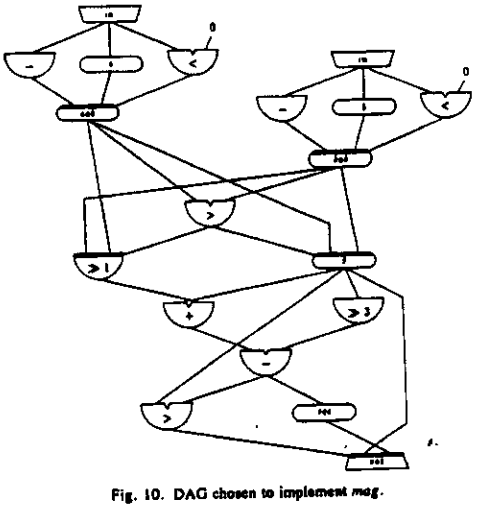


Fig. 10. DAG chosen to implement *mag*.

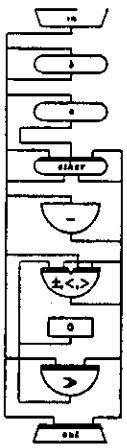


Fig. 11. Bitslice for *mag* datapath.

maximum amount of sharing of wiring tracks for busses. In the *mag* example, the ordering allows two short busses to share the track second from the left, and then keeps the rightmost bus as short as possible. The actual output isn't particularly interesting to look at. The datapath description file lists the resources in order, along with their particular capabilities and I/O multiplexers, and then the connections between busses and functional units. The controller description file lists for each block, cycle-by-cycle, what values are gated to and from what busses, and what the functional units do.

VI. Results

It is hard to characterize theoretically what a "typical" input program looks like, so theoretical results about Flamel's algorithms are hard to come by. For this reason, a prototype implementation has been produced.

How should the quality of Flamel's output be assessed? One way is to compare it with the output of other silicon compilers. Nearly all of the previous work similar to Flamel is not yet at the stage where results are being reported. The main exception to this lack of results is the work at CMU [1], [2]. Unfortunately, their test cases are things like general-purpose computer architectures; Flamel would certainly do rather poorly on such tests, since no attention has been paid to the details that are important for such tests. In particular, the poor support for bit manipulation in Flamel is fatal. Also, the programs for emulating general-purpose computers tend to be uninteresting from the point of view of global optimization, since the instruction fetch-decode-execute loop takes only a few cycles.

One thing is certain: a chip produced by Flamel better run considerably faster than an off-the-shelf microprocessor running the input program, because the microprocessor will probably be cheaper, owing to economy-of-scale. At the time this research was begun, the Motorola MC68000 was a typical microprocessor, and the availability of a good Pascal compiler targeted to the M68000 made it a natural microprocessor to compare with.

TABLE I
Speedups: MC68000 External Clock Cycles/Flamel Chip Cycles

	Most Serial	Basic Blocks	Most Parallel
minimum	8.3	9.9	22.7
maximum	14.5	28.4	217.2
median	11.1	15.1	43.5
average	11.7	15.9	67.8

Each of the test programs was compiled with pc68, a Stanford-produced Pascal compiler that generates good code. It incorporates Chow's global optimizer [22]. The assembly listing of the generated code for each test was examined to calculate how many external clock periods an MC68000 would use in executing the program, given the same basic block frequencies that Flamel used in calculating execution time.

Some statistics about the ratio of MC68000 cycles to Flamel cycles are shown in Table I. The numbers show how many times faster a Flamel chip would be compared to an MC68000 if they were both clocked at the same rate. Of course, the MC68000 could probably be clocked faster than a Flamel chip because it was hand-tuned by experts. But a Flamel cycle need only be long enough to do a bus transfer followed by an add, so hand tuning could probably make a Flamel chip operate at a rate comparable to the MC68000. Even if the Flamel chip cycle were twice as long as that of the MC68000, the speedups are still good.

There were 15 test programs, chosen to be typical of code fragments that might appear in general-purpose software. The programs do things like bubble-sort, convert strings of digit characters to numbers, count words, implement a finite impulse response filter, and manage a hash table. The hash table program was the longest, about 70 lines. Complete listings can be found in Trickey [15].

The "most serial" column of Table I is what Flamel produced when asked to put only one noncopy operation in each cycle and not to do any block-level transformations. The "basic blocks" column is the most parallel possible implementation without doing block-level transformations. The "most parallel" column gives the fastest implementation Flamel could find.

One can see from the "most serial" column that an order of magnitude speedup is due simply to the fact that a Flamel chip doesn't have to fetch instructions and data from memory, and doesn't have to do address arithmetic. But the difference between the "most serial" and "most parallel" columns shows that when all of Flamel's techniques are brought into play, there are an average of 4.8 noncopy operations being done in parallel. The "most serial" implementation is similar to what would be achieved using a microcoded standard processor. An experiment

TABLE II
DATA FOR SOME BENCHMARK IMPLEMENTATIONS

Test	Cycles	Datapath Area ($\lambda^2 \times 10^6$)	Controller Area ($\lambda^2 \times 10^6$)	Compilation Time (s)
presum	58	.650	.068	4.7
	30	1.011	.082	4.7
	23	1.272	.100	15.5
	12	1.462	.138	14.5
	6	8.342	.098	18.5
	5	8.342	.098	21.1
mag	19	.759	.108	18.4
	13	.878	.104	19.0
	9	.897	.092	18.1
mmult	204	1.063	.310	21.2
	100	2.029	.165	16.9
	68	2.298	.167	17.9
	66	2.514	1.287	463.3
hash	444	8.715	1.042	106.6
	248	10.171	1.087	103.1
	107	18.729	3.644	617.4

reported in Trickey [15] showed that a microcoded RISC architecture (MIPS [23]) would run in about as many cycles as a "most serial" Flamel chip.

One of the most interesting results of these tests is that there is hardly any parallelism available if no block-level transforms are done. Some previous attempts to parallelize ordinary programs have met with discouraging results, but those attempts didn't look for parallelism at more than a basic block at a time.

Let us now examine four of the benchmarks in detail, to see what time/area tradeoffs can be achieved. Table II contains data about some of the implementations that Flamel finds with different settings of the cost bound. *Mmult* and *mag* were given in Figs. 3 and 9, *presum* does the parallel prefix sum computation discussed at the end of Section IV, and *hash* is an expanded version of the hash table procedure in Kernighan and Plauger [24]. The areas are given in units of mega-λ^2, where λ is half the minimum feature size. The nMOS cells used in the area calculations are similar to those used in Stanford's Information Systems Laboratory [25].

All of the *presum* implementations are given. It can be seen that Flamel achieves a wide range of time/area tradeoffs. The slowest implementation has a lengthened schedule in an attempt to meet the cost bound, and the fastest implementation uses a fully unrolled loop, height reduced. The intermediate cases had partial degrees of block transforms, loop unrolling, and height reduction. The table shows only some of the implementations for the other tests. The *mag* designs are similar to or better than those found by MacPitts, but MacPitts requires that the user change the input to get the different designs [20]. The author implemented *mmult* by hand; it was several times smaller than the comparable Flamel design, but the one implementation took several months to complete.

Another thing to notice from Table II is that the compilation time is reasonable. The times are given in CPU seconds on a VAX 11/780. Furthermore, Flamel would run much faster with more sophisticated data structures.

The areas estimated for the most parallel versions of the test programs were almost all smaller than today's VLSI microprocessors. The exceptions were programs that could be made faster by making large, many-ported register files to implement some arrays. Much of the area required by *hash* is due to seven arrays, with a total of 343 entries. In most of those cases, tapped shift registers could have achieved comparable speedups at a fraction of the cost. A future version of Flamel should know how to use shift registers. If larger programs than the hash table manipulator were tried, the datapath area probably wouldn't grow too much unless more and larger arrays were used. Larger programs probably won't be able to have much more than the approximately five things going on at once that Flamel found in the test programs. On the other hand, the controller area would start to dominate if much bigger programs were tried. Flamel has to pay more attention to controller area than it does.

Some tests were run to see which of Flamel's techniques were most important in getting best speedup over the most serial implementation. Here are the average percentages of total speedup achieved when certain transformations were omitted:

- block transforms omitted: 26%,
- loop unrolling omitted: 40%,
- height reduction omitted: 71%.

See Trickey [15] for further statistics, including statistics about the efficacy of the cleanup transformations.

VII. Conclusions

The results of running Flamel on test cases are encouraging. Respectable speedups can be found when a datapath with multiple functional units is used rather than a more standard architecture with a single ALU. Also, Flamel has shown itself capable of producing a wide range of implementations, so a user can quickly get many designs with different performance-area tradeoffs. While the emphasis in this work has been on custom VLSI for the implementation target, most of the techniques would apply equally well to gate array and standard cell implementations.

Compile time is the best time to schedule, when it works. This argument has been put forward by others [26], [27] as an answer to dataflow machines. A dataflow machine can be viewed as determining at execution time the operations that can be done in parallel, and scheduling them on available processing units. It also has to schedule the transfer of data to and from memory. All of this work is expensive in hardware. Furthermore, it appears to slow the rate at which individual processing elements can be supplied with operands. The result is that a lot of parallelism needs to be found in the executing programs before the dataflow machine beats a conventional processor.

Some parallelism can only be detected at execution time. For example, two array subscripts might depend on input data, so that one has to wait until execution time to determine whether certain calculations can be done simul-

taneously. It remains to be an. red how much of this type of parallelism can be expected, but the results reported elsewhere [28], [29] show that at least a 5–10 times speedup can be expected from scientific programs, and the results of this work show that a 2–8 times speedup can be expected from general-purpose software.

Some of the contributions of this work are as follows.

- It has been shown that one easy-to-write behavior specification can generate a wide range of time/area tradeoff implementations, with a bare minimum of effort required from the user. Up to now, general-purpose digital system design has been regarded as a domain where an expert is needed to either choose the architecture or at least to guide the choice.
- An adequate set of global (block-level) and local (DAG-level) transformations has been identified, along with a controlled method for applying them. It is easy to suggest optimizations to apply during silicon compilation—after all, most of them have long been known to the software compiler community. Others have done this, but none have given feasible methods for deciding which to apply and when. For instance, Kowalski's system incorporates a number of global optimizations, but the user has to decide by hand which to apply and when [2]. Flamel's dacon-choosing procedure is a step in that direction. Also, the optimizations described here have all been found to be useful in practice, since they were added on an as-needed basis.
- This is the first work to do extensive global optimizations at all; yet the results show that the global optimizations are crucial for extracting much parallelism from ordinary programs. (Fisher's trace scheduling [29] achieves a similar effect by scheduling at a global level; but he intends it for a fixed architecture).
- A height-reduction technique has been developed to work on DAG's rather than simply trees. The combination of the height-reduction technique and the other transforms yields results that subsume a number of specialized techniques used in vectorizing compilers.
- The prototype implementation revealed useful statistics on the relative cost and effectiveness of the proposed optimization techniques. For instance, it showed what size and speed might be expected from a Flamel-designed chip for various pieces of code. Also, the fact that Flamel runs reasonably quickly shows that a moderate amount of search (in dacon-choosing) is not out of the question.

There is a lost of room for improvement in Flamel. Trickey [15] discusses some ideas in a number of areas that deserve further investigation: 1) more than one procedure in the input program; 2) handling the timing of communication with the outside world; 3) pipelining, both at the operation level and the block level; 4) special implementations for arrays; 5) incorporating controller area into the costs.

REFERENCES

[1] A. C. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Lieve, and J. Kim, "The CMU design automation system," in *ACM IEEE 16th Des automation Conf. Proc.* (San Diego), 1979, pp. 73–80.

[2] T. J. Kowalski, "The VLSI design automation assistant: A knowledge-based expert system," Ph.D. thesis, Carnegie Mellon Univ., 1984. Available as CMU Technical Report CMUCAD-84-29.

[3] T. J. Kowalski and D. E. Thomas, "The VLSI design automation assistant: Prototype system," in *ACM IEEE 20th Design Automation Conf. Proc.* (Miami), 1983, pp. 479–483.

[4] K. Palem, D. S. Fussel, and A. J. Welch, "High-level optimization in a silicon compiler," Tech. Rep. TR-215, Dept. of Computer Sciences, University of Texas, Austin, TX, 1982.

[5] D. Knapp, J. Granacki, and A. C. Parker, "An expert synthesis system," in *Proc. of the Int. Conf. on Computer Aided Design*, ACM and IEEE, 1983, pp. 419–424.

[6] T. J. Kowalski, D. J. Geiger, W. H. Wolf, and W. Fichtner, "The VLSI design automation assistant: From algorithms to silicon," *IEEE Design and Test*, pp. 33–43, Aug. 1985.

[7] P. E. Agre, "Designing a high-level silicon compiler," in *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers* (Port Chester, NY), 1983, p. 413.

[8] J. M. Siskind, J. R. Southard, and K. W. Crouch, "Generating custom high performance VLSI designs from succinct algorithmic descriptions," in *Proc. Conf. on Advanced Research in VLSI*, (MIT), Jan. 1982, pp. 28–39.

[9] H. E. Shrobe, "The data path generator," in *Proc. Conf. on Advanced Research in VLSI*, (MIT), Jan. 1982, pp. 175–181.

[10] J. L. Hennessey, "SLIM: A simulation and implementation language for VLSI microcode," *Lambda*, pp. 20–28, Apr. 1981.

[11] A. R. Karlin, H. Trickey, and J. D. Ullman, "Experience with a regular expression compiler," in *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers* (Port Chester, NY), 1983, pp. 656–665.

[12] A. V. Aho and J. D. Ullman, *Principles of Compiler Design*. Reading, MA: Addison-Wesley, 1977.

[13] M. C. McFarland, "The VT: A database for automated digital design," Tech. Rep. DRC-01-4-80, Design Research Center, Carnegie Mellon University, 1978.

[14] G. S. Tjaden and M. J. Flynn, "Detection and parallel execution of independent instructions," *IEEE Trans. Comput.*, vol. C-19, no. 10, pp. 889–895, 1970.

[15] H. Trickey, "Compiling Pascal programs into silicon," Ph.D. thesis, Stanford Univ., July 1985. Stanford Computer Science Report STAN-CS-85-1059.

[16] D. J. Kuck, *The Structure of Computers and Computations*, vol. 1. New York: Wiley, 1978.

[17] P. M. Kogge and H. S. Stone, "A parallel algorithm for efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.*, vol. C-22, pp. 786–792, Aug. 1973.

[18] U. Banerjee, D. Gajski, and D. J. Kuck, "Array machine control units for loops containing IFs," in *Proc. 1980 IEEE Int. Conf. on Parallel Processing*, 1980, pp. 23–36.

[19] C. Y. Hitchcock III and D. E. Thomas, "A method of automatic data path synthesis," in *ACM IEEE 20th Design Automation Conf. Proc.* (Miami), 1983, pp. 484–489.

[20] J. R. Southard, "MacPitts: An approach to silicon compilation," *Computer*, vol. 16, pp. 74–82, Dec. 1983.

[21] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.

[22] F. Chow, "A portable machine-independent global optimizer—Design and measurements," Ph.D. thesis, Stanford Univ., Dec. 1983.

[23] S. Przybylski, T. Gross, J. Hennessy, N. Jouppi, and C. Rowen, "Organization and VLSI implementation of MIPS," *J. of VLSI and Computer Systems*, vol. 1, Spring 1984. Available as Tech. Rep. 83-259, CSL, Stanford.

[24] B. W. Kernighan and P. J. Plauger, *Software Tools in Pascal*. Reading, MA: Addison-Wesley, 1981.

[25] J. Newkirk, R. Mathews, J. Redford, and C. Burns, "Stanford nMOS cell library," Tech. Rep. 001, Information Systems Laboratory, Stanford Univ., 1981.

[26] D. D. Gajski, D. A. Padua, D. J. Kuck, and R. H. Kuhn, "A second opinion on data flow machines," *Computer*, vol. 15, pp. 58–69, Feb. 1982.

[27] J. A. Fisher, J. R. Ellis, J. C. Ruttenberg, and A. Nicolau, "Parallel processing: A smart compiler and a dumb machine," in *Proc. ACM SIGPLAN Symp. on Compiler Construction* (Montreal, Canada), 1984, pp. 37–47.

[28] D. J. Kuck, "Measurements of parallelism in ordinary FORTRAN programs," *Computer*, vol. 7, pp. 37–46, Jan. 1974.
[29] J. A. Fisher, "Trace scheduling: A technique for global microcode compaction," *IEEE Trans. Comput.*, vol. C-30, pp. 478–490, July 1981.

Howard W. Trickey (S'76–M'85) received the B.A.Sc. and M.A.Sc. degrees from the University of Toronto in 1978 and 1980, and the Ph.D. degree from Stanford University in 1985.

He is now a Member of Technical Staff at AT&T Bell Laboratories in Murray Hill, NJ. His research interests include hardware synthesis, compilation for parallel machines, and text processing and illustration tools.

Def #5

The CMU Design Automation System

An Example of Automated Data Path Design

A. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive, J. Kim

Carnegie-Mellon University
Departments of Electrical Engineering and Computer Science
Pittsburgh, Pennsylvania 15213

Abstract

This paper illustrates the methodology of the CMU Design Automation System by presenting an automated design of the PDP-8/E data paths from a functional description. This automated design (using synthesis techniques) is compared both to DEC's implementation and the Intersil single chip implementation.

1. Introduction

As it is becoming possible to integrate larger numbers of logic components on a single chip, the need for more powerful design aids is becoming apparent. Indeed, these aids must be capable of supporting a designer from the system level of design down to the mask level. In this way the systems level designer can become more aware of the implications of higher-level design tradeoffs on implementation properties such as silicon area, power consumption, testability, and speed, and be able to make more timely use of new technologies. The ultimate goal of the Carnegie-Mellon University Design Automation (CMU-DA) System [12] is to provide a technology-relative, structured-design aid to help the hardware designer explore a larger number of possible design implementations. Inputs to the system are a behavioral description of the system to be designed, an objective function which specifies the user's optimization criteria, and a data base specifying the hardware components available to the design system.

The CMU-DA system differs from other design automation systems because the input design description is a functional specification. Such a specification provides a model that, while accurately characterizing the input-output behavior desired for the implementation, does not necessarily specify its internal structure. The system software collectively performs the synthesis function by transforming the input functional description into a structural description. The design process involves binding implementation decisions in a top-down manner as a design proceeds through the design system. More structural decisions are made at each level until a complete hardware specification is obtained, with the most influential design trade offs being performed first in order to cut down the design search space.

The purpose of this paper is to illustrate the methodology of the CMU-DA system. The results given here are worst case - many optimizations which are straightforward have not been implemented yet; research is in progress on others. The design of the data part of a DEC PDP-8/E [5] from the ISP level through to a TTL and standard cell design will be discussed. Only the subset of the full DA system which is presently implemented has been used for this example. The components are shown in Figure 1.

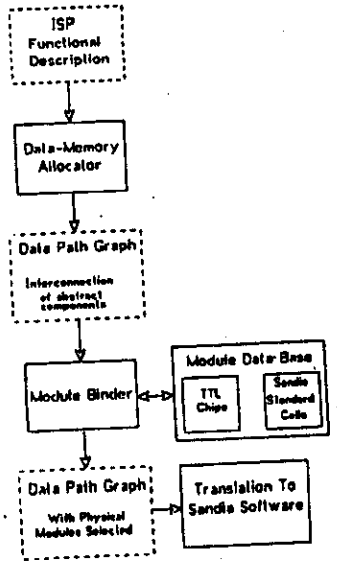


Figure 1: The CMU Design System

The PDP-8/E is first functionally described using the ISP language [1]. A data-memory allocator [7] is used to generate the structure of the data paths from the functional description. This allocation is in terms of abstract logic components. The next step in the design is to bind physical modules to the abstract design from a module database [9]. This step provides the system with the capability of designing relative to new technologies. This binding will be illustrated both in terms of TTL chips and the CMOS standard cells in the Sandia design system [10]. The standard cell output of the CMU-DA system is then translated for input to the Sandia design system. At this point, chip area can be calculated and detailed timing information can be gathered using SALOGS [3].

The paper will conclude by comparing these two alternative designs to commercial implementations.

This research is supported in part by NSF Grant MCS77-09730 and Army Research Office Grants DAAG29-76-G-0224 and DAAG29-78-G-0070.

RCL000199

2. The PDP-8/E Example

As the CMU-DA system has evolved, more complex design examples have been used to observe its performance. The use of the PDP-8/E is a major step in two ways: 1) it represents about a five fold increase in the circuit complexity over previously reported example designs [8], and 2) it is a commercially available system that has been implemented in various technologies by different manufacturers over several years. Thus it is possible to compare a non-trivial automated design to designs done by several designers with different logic components.

The PDP-8/E is one model in a family of computers having nearly identical ISP descriptions. (That is, they implement nearly the same instruction set, with different hardware structures). A portion of the ISP description used by the automated design system is shown in Figure 2.

The description begins by declaring the memory and processor state. A 13 bit link-accumulator (lac) register is defined but can alternately be accessed as the one bit link (L) and 12 bit accumulator (Ac). Next, instruction interpretation is defined using the declared memory and registers. After the instruction fetch and increment of the program counter in the instruction interpretation section, instruction execution (exec()) is called. Illustrated here are the six memory reference instructions. Not shown but implemented in the automated designs are the effective address calculation, input/output instructions, and the operate microinstructions. Note that the Mb←Mp[Pc] instruction fetches the location pointed to by the Pc and places it in the Mb register. No mention need be made of transfer of the current Pc to a memory address register, which is a part of the hardware structure.

The ISP description [1] is compiled into a representation which is machine readable by the data-memory allocator. The next sections will discuss the data-memory allocation (where an abstract data path is synthesized), a module binder that illustrates how the CMU-DA system can design relative to new technologies, and a translator to Sandia's SALOGS simulator [10].

3. The Allocation Process

The data-memory allocators perform a mapping function from the algorithmic (ISP) description to the data-path part of the hardware implementation, which is called a data-path graph. The data part consists of the data-storage elements, data operators, and data paths necessary to implement the operations specified in the algorithmic description. Due to the characteristics of the ISP language this mapping may be multi-valued in either direction, rather than a simple one-to-one translation.

The PDP-8/E design described here was produced by a data-memory allocator which uses the distributed-logic design style. This style of (or approach to) design encompasses design with small and medium scale integration components. As pointed out earlier, the allocator itself is technology relative and the mapping onto specific integrated circuit packages is performed by a separate module binder program. The process referred to as allocation throughout the remainder of this paper is a synthesis of logic using generic logic elements; data paths, operators, registers, and multiplexers, all of any bit width.

The procedure used by the allocator might be compared to a two-pass compilation. The first pass may be considered a syntax or feasibility check. The allocator inputs a parsed ISP description, constructs data structures analogous in function to symbol tables, and enforces constraints necessary to insure that the data-storage locations, logical mappings, and

```
pdp8 :=
   BEGIN

! The basic PDP-8 instruction set (without extended arithmetic
! element) is implemented.  No I/O devices are included in the
! description.

! MP State

    Mp[#0:#7777]<0:11>,                     ! Main memory
    Mb<0:11>,                               ! Memory buffer

! PC State

    lac<0:12>,                              ! Link  ac register
      L<>       := lac<0>,                  ! Link bit
      Ac<0:11>  := lac<1:12>,               ! accumulator
    Pc<0:11>,                               ! Program counter
    last.pc<0:11>,

! Instruction Interpretation

    start :=
        BEGIN
        go = 1 NEXT
        run()
        END,

    run\instruction.interpretation :=
        BEGIN
        IF go =>
            BEGIN
            Mb = Mp[Pc]; !w:1.pc = Pc NEXT
            Pc = Pc + 1 NEXT
            exec() NEXT
            IF interrupt.enable AND interrupt.request =>
                BEGIN
                Mp[0] = Pc NEXT
                Pc = 1
                END NEXT
            RESTART run
            END
        END,

! Instruction Execution

    exec\instruction.execution :=
        BEGIN
        ir = Mb<0:2> NEXT
        IF (ir GEQ #3) AND (ir LEQ #5) => ea() NEXT
        IF ir LEQ #2 => Mb = Mp[ea()] NEXT
        DECODE ir =>
            BEGIN
            #0 := and.:= Ac = Ac AND Mb,        ! And
            #1 := tad := lac = lac + Mb,        ! ADD (TC)
            #2 := isz := BEGIN                  ! Increment and
                                                ! skip if zero
                         Mb = Mb + 1 NEXT
                         IF Mb EQL 0 => Pc = Pc + 1
                         END,
            #3 := dca := BEGIN                  ! Deposit and
                         Mb = Ac NEXT           ! clear Ac
                         Ac = 0
                         END,
            #4 := jms := BEGIN                  ! Jump to
                         Mb = Pc NEXT           ! subroutine
                         Pc = ea + 1
                         END,
            #5 := jmp := Pc = ea,               ! Jump
            #6 := iot(),                        ! I/O execution
            #7 := opr()                         ! Operate
                                                ! microinst.
            END NEXT
        IF (ir GEQ #2) AND (ir LEQ #4) => Mp[ea] = Mb
        END
    END         ! End of description
```

Figure 2: ISP Description of the PDP-8/E

RCL000200

input/output interface characteristics specified in the description can be implemented in hardware. If no errors are encountered, it proceeds to allocate the basic data-storage structures called for in the description, and any additional data paths, storage, and operators necessary to implement variable-accessing schemes described by ISP. The second pass may be considered as the semantic phase, with the activity of code generation replaced by the allocation of data paths, operators, and additional storage as needed to implement the actions described in the ISP description. Parallelism analysis is performed at several levels to warn the user of error conditions and determine constraints relating to optimization of hardware. The allocation is then completed by the addition of multiplexing where required.

However, allocation differs from compilation in that in a compilation one is concerned with implementing the specified data operations on a fixed data part whose capabilities are known *a priori*. In allocation, the allocator must be able to recall and utilize the capabilities of a data part which is being dynamically created. The allocator thus works from the inside out, first creating the data storage and access structures, and then adding the necessary data paths and operators to perform the described data operations. Finally, the output of the allocator is a non-planar directed graph, rather than a linear list of compiled instructions.

The first version of the allocator is experimental, and it performs only minor optimizations on the allocated hardware. It has been designed to investigate the feasibility of performing the mapping from ISP to hardware, the types of data structures needed for allocation, and areas where optimizations are possible in future, more sophisticated allocators.

The allocator has been designed as a possible skeletal structure for future allocators in order to standardize input/output formats and data structures.

4. Performance of the Data-Memory Allocator

The allocator program was run using the ISP description of the PDP-8/E and the resultant data paths are shown in Figure 3. A binding of modules was done by hand to compare the results of the allocator to the original DEC design (Figure 4) [5].

It is difficult to compare the automated PDP-8/E data-path design with the original DEC design for three reasons. First, the ISP description input to the allocator declares as registers some values the PDP-8/E uses but never stores explicitly in registers, such as the effective address. These show up as registers in the allocator's design. Second, the allocator designs distributed logic, and the DEC design was done in the central-accumulator design style. (That is, this allocator does not contain the design rules for large scale collapsing of the data paths into a central-accumulator style of design [13]). Third, the DEC design has assumed a boundary between the control and data-memory parts of the design, but the boundary is different from that imposed on the allocator by the ISP description. Thus some links, flags, and registers which must be declared explicitly in the ISP description are part of the control in the DEC design.

The main reason for the difference in design seen from the block diagram level of Figures 3 and 4 is that the design styles are different. The multiplexing is used in different ways. In the DEC version, the operators are shared, and are used to provide no-op paths from one register to another. In the CMU version, only registers are shared and use multiplexed inputs. The ISP language is partially the source of this disparity. In ISP, the user can repeatedly use register A as a destination from various sources. However, the expressions A+B and C+D do not imply nor discount a single adder. Other differences in the design include the use of multiplexers for shifting in the DEC design, and use of true/complement 0/1 chips for creating complements. "Oring" of the MQ and AC registers in the DEC version is done within the multiplexing hardware. Constants are often created in one place and gated over already existent data paths to the registers. In the CMU version, these constants are multiplexed at the register inputs.

In spite of these differences, estimates of chip count indicate that the allocator produces a path graph which would require 39% more integrated circuit chips that the DEC designers used for the data paths and registers. These estimates were done by hand to gauge the performance of the allocator; an automatic binding is discussed in the next section. These estimates were made using the same 1970 technology chip set the DEC designers had to deal with. The 39% excess hardware can be found in multiplexors which connect the registers, the extra registers declared in the ISP description, and duplicated operators like increment and add. Much of this excess can be attributed to the lack of optimization capability in the allocator algorithm. Future, more sophisticated allocation algorithms, coupled with the capability for high level optimization [12] available in the complete CMU-DA system will be able to significantly improve the data part design.

Further analysis of this design is in progress and includes a manual implementation of the control part. Comparisons of the DEC and CMU data-path speeds will then be possible.

5. Module Binding

The module binding phase of the CMU-DA system employs the Module Data Base System (MDBS) and follows the data-memory allocation step. It has the task of translating the abstract links, memories, registers, and operations in a data-path graph into a design using physically realizable modules. A second pass of module binding will occur after control allocation.

At present the module binding portion of the design system is primarily a research tool that will be used to investigate automated module binding. A goal of this research is to model the module binding problem sufficiently to generalize this part of the CMU-DA system to handle a wide range of module types from LSI chips through Standard Cells. The implemented portions of MDBS were used to assist a designer in binding data-part modules to the CMU PDP-8/E data-paths produced by the Data-Memory Allocator described in Section 4. The results of the data-part module binding are compared to the DEC PDP-8/E design in Section 7. A similar comparison will be made to the standard-cell binding after those results are presented in Section 8.

6. Organization of MDBS

MDBS consists of four sections shown in Figure 5: an I/O section that is responsible for translating between the internal and external forms of the path graphs a Module Data Base access mechanism; a command language interface; and the module binding mechanism.

The input/output section is the interface to other parts of the CMU-DA system. The path graph, generated by an allocator is placed in internal form for processing. The output file has the same format as the input path graph (with module binding information appended) and can be reread by the input section for additional processing.

The Module Data Base is a hierarchical data base that is distributed in various ASCII files. The highest level of the data base is the index which is read automatically during system initialization. The index contains pointers to all defined design-style sets, each of which is a collection of module sets appropriate for a given design style. A design style set file contains pointers to the actual module set information (the "Data Book") and summary information (typical cost, speed,

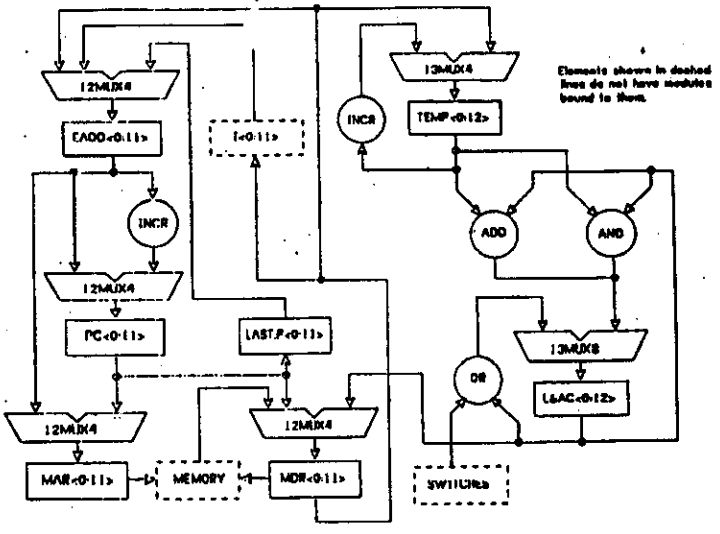


Figure 3: Allocator Generated PDP-8/E Data Paths

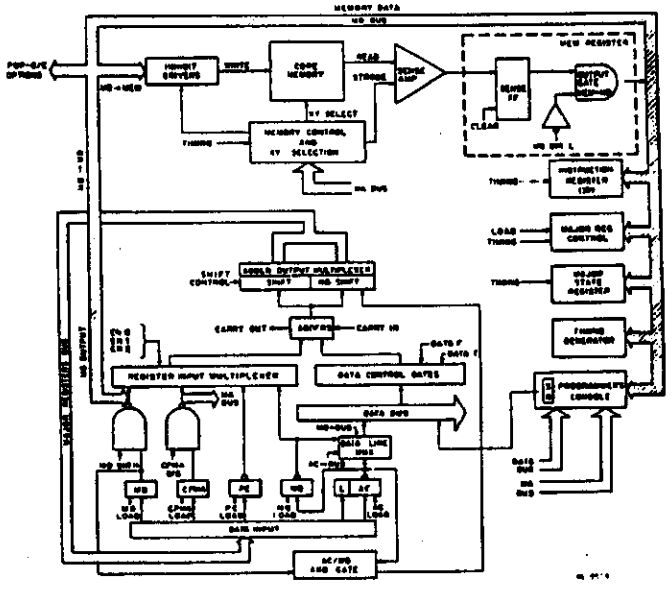


Figure 4: IXC PDP-8/E Data Paths

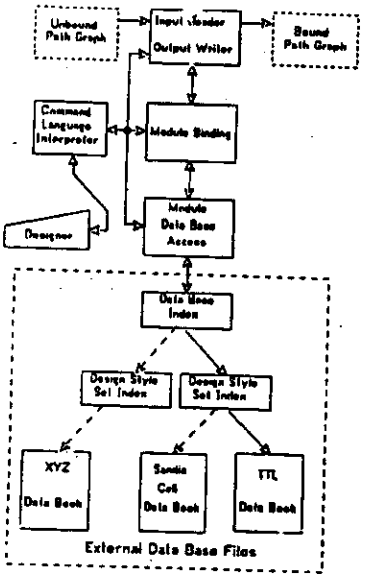


Figure 5: Organization of the Module Data Base System

load, and drive capabilities of modules in the set). Access to data base information during module binding occurs frequently and utilizes various levels of detail from basic type matching through specific delay and timing characteristics.

The Command Language Interpreter (CLI) is the experimenter/designer's interface to the module binding system. The CLI may be used only to select a file to be processed and direct the disposition of the resulting bound path-graph. The CLI also provides a number of tools to inspect the path-graph, modify the graph structure and bind designer selected modules to specific nodes of the graph.

The module binder applies transformations to both the path graph and module functions in order to match the desired behavior with the available building blocks. The graph transformations are localized decompositions or combinations of nodes that preserve the specific behavior.[7] Module transformations are primarily combinations of nodes since modules cannot be physically decomposed. However, multifunction modules such as shift registers and ALUs may require partitioning of the non-conflicting functions of the same module onto separate nodes of the graph.

The most prevalent type of graph transform is localized to a single node or several connected nodes of a similar type. Registers are usually decomposed into nodes of smaller bit width. Logical operators are usually modified by reduction to a canonical form, then synthesized with available module operations. Multiplexors and demultiplexors are frequently transformed from a single level to a multilevel form. Arithmetic operators (particularly the signed and complement arithmetic modes) require algorithmic decomposition.

A more complex type of transformation involves the combination of nodes of different types into single nodes capable of multiple functions. Shift operators and special purpose arithmetic operators (increment, decrement, and clear) are generally combined with register functions in available module sets. These transformations often provide significant reductions in the graph complexity by elimination of constants and reduction of multiplexor size.

Operator node transformations primarily involve the application of axioms and identities to combine available modules into an aggregate that performs a desired function. Boolean identities and DeMorgan's Theorem will direct logical synthesis. Arithmetic mode transformations (for unsigned, signed-magnitude, two's complement, and one's complement) will be utilized to synthesize required modes from available modes.

For cases where single transformations on either the graph or the modules do not provide the desired match, an iterative approach will be utilized. The graph and the module transforms will be alternately applied until one or more matches are found, a cycle is detected in the transformations, no further gain is detected by applying transforms, or one of the system constraints (speed, cost, etc.) is violated by the resulting implementation.

A central goal of the CMU-DA system is to produce designs that have been optimized toward the designer's objectives and fall within the external constraints. Module binding is the first operation in the design system that attaches actual costs to the implementation and has specific speed, delay, and power information available. Therefore, evaluation of the bound design must be performed to insure compliance with the constraints. Critical constraints (i.e. constraints which the designs *must* meet) will be dynamically estimated by projecting the final value based on an extrapolation from the number of nodes remaining to be bound and the accumulated value for the nodes already bound; a true evaluation of the fully bound design will be made as a final pass to insure compliance with the constraints. The dynamic evaluation will be used to select between functionally identical module choices with different performance parameters.

7. PDP8 Baseline Example

A partial module binding of the PDP-8/E was done with the available pieces of MDIS, which contain limited transform capabilities.

The module binding section contains a set of primitive operations that can modify the path graph structure (but not its behavior). The operations allow register nodes to be split at bit boundaries, operator nodes to be joined into single nodes with wider bit widths, nodes to be inserted, and nodes to be deleted. These operations allow the path graph nodes to be transformed to conform with the structure of available modules. As the MDIS becomes more fully implemented, these primitive operations will be used to build larger scale path graph transforms that can be applied automatically.

The existing system aids the designer in producing correctly bound path graphs by the strict enforcement of rules concerning application of the structural modification primitives. These operations are restricted to minimize the possibility of modifying the behavior of the path graph. Some examples of these rules are:

- Nodes may not be deleted while connected to more than one link.
- Links may not be deleted while joining two nodes.
- Half links (i.e., signals to external sources) may not be deleted.
- Operator nodes may be joined only if they are of the same type.

RCL000203

A different but equally important kind of assistance is provided to allow the designer to display any aspect of the binding process. For example, the designer may display one module in the data base, all modules providing a specific function, or the entire data book.

The module choices were made by the designer using information from the path graph and a small TTL module data base from the distributed Design Style Set. The purpose of binding the data part with the existing system is to contrast the module selection with a hand designed PDP-8/E. This example provides a worst-case from which to judge the performance of MOBS as more capabilities are added.

Figure 6 was generated by the MOBS. The registers (named variables), operators, and multiplexors are identified in the "Comp." (Component) column. The "Device" column lists the name of the selected package. "Mods." lists the number of modules required to implement the component function. The term "module" refers to a separate functional unit in this context. There may be several modules contained in one package. "Pkgs." lists the number of packages required for each function. "Gates" is an estimate of the equivalent number of logic gates to implement the function. The percentage of the total gates required is listed in the "% Total Gates" column. The cost of each function implementation is computed as the basic cost for the number of packages required plus an overhead mounting cost of $3.00 per package. The percentage of the total cost attributed to each function is listed in the "% Total Cost" column.

Comp.	Device	Mods.	Pkgs.	Gates	%Total Gates	Cost	%Total Cost
ENDD	SN74174	2	2	78	3.98	7.78	2.58
INC	SN74194	3	3	105	5.36	11.67	3.87
LAST.P	SN74174	2	2	78	3.98	7.78	2.58
PC	SN74161	3	3	264	16.88	11.67	3.87
MQR	SN74174	2	2	78	3.98	7.78	2.58
MAR	SN74174	2	2	78	3.98	7.78	2.58
TEMP<8>	SN7474	1	1	6	0.31	3.35	1.11
TEMP	SN74174	2	2	78	3.98	7.78	2.58
EQL	SN7485	3	3	93	4.87	11.37	3.77
EQL	SN7485	3	3	93	4.87	11.37	3.77
INCR	SN7483	3	3	108	5.68	10.77	3.57
INCR	SN7483	3	3	108	5.68	10.77	3.57
AND	SN7408	12	3	12	0.83	9.69	3.18
OR	SN7432	12	3	12	0.83	9.78	3.23
MUX	SN7403	4	4	144	7.34	14.36	4.78
13MUX8	SN74151	13	13	156	8.17	48.67	15.48
12MUX4	SN74153	12	8	96	5.03	21.56	7.14
12MUX4	SN74153	12	8	96	5.03	21.56	7.14
12MUX4	SN74153	12	8	96	5.03	21.56	7.14
12MUX4	SN74153	12	8	96	5.03	21.56	7.14
13MUX4	SN74153	13	7	104	5.45	25.13	8.33
Totals		131	83	1909	100.00	301.56	100.00

Component Class	% Gates	% Cost
Registers	36.48	19.17
Operators	29.88	23.49
Multiplexors	33.74	52.37

Figure 6: Module Utilization For PDP-8/E Data Part

The choice of registers differed from the hand module binding in a few locations. The hand bound PDP-8/E used SN74194s (four bit universal shift registers) exclusively while the MOBS chose SN74174s (six bit D registers) when there was no requirement for the added shift capability. The Program Counter (PC) register was selected as three SN74161s (universal counters) based on the INC flag associated with the path graph node. The Accumulator (LAC) was first split into a one bit node and a twelve bit node, then the twelve bit part was allocated as three SN74194s. This is the same allocation that was done by hand. However, the choice only satisfies the shift ration flags (LSHFT and RSHFT). The increment requirement (INC) has effectively been partitioned out and must be bound separately.

The package count was 30% higher for the MOBS selection than for the DEC implementation (64 packages for DEC vs. 83 packages for MOBS). This agrees closely with the results obtained by selecting modules strictly by hand (refer to Section 4). The 30% difference is attributable to the different design styles used and the allocator's implementation of the design. Comparing the total cost of modules for the DEC implementation and the MOBS implementation (Figure 6), it is found that the costs also are 30% higher for the automated implementation, while the number of equivalent gates is 65% higher for the automated implementation. This indicates that MOBS chose modules with a higher level of integration than DEC did.

A comparison of the percentage of equivalent gates and the percentage of cost accumulated in three functional classes (registers, operators, and multiplexors) indicates surprisingly uniform comparisons. The percentage of both gates and cost is higher for registers in the MOBS implementation than in the DEC implementation. This trend is expected since the DEC PDP-8/E uses a central accumulator design style. Also, the slightly lower percentages for gates and costs in the operator class is reasonable for the DEC implementation. The most surprising comparison is the near identical percentage of gates devoted to data path routing (i.e. multiplexors) in the two designs implemented in different design styles. It would be expected that the central accumulator style would utilize more data path routing than the distributed style. This apparent anomaly is a clue to the area where the module binding can make local improvements in a path graph for distributed designs. By utilizing functions intrinsic to certain modules (such as the CLEAR on registers), constants and their associated data paths can be eliminated and improve the cost of implementing a design.

The TTL module binding using MOBS compares favorably with the DEC implementation and previous hand module bindings of the automated path graph. It is expected that much improvement in the package count (and the cost) is forthcoming as transforms and evaluation techniques are implemented in MOBS. However, TTL module binding is just one objective of a generalized design system. The following section discusses an approach to binding CMOS standard cells to a design with the objective of being able to automate and produce LSI designs.

8. Standard Cell Generation

The Sandia standard cell library [11] can also be used as physical modules to implement the automated PDP-8/E data part design. Then, using the Sandia software package [4, 10], it is possible to produce a simulation, insert faults, and perform automated cell placement and IC mask generation for a CMOS LSI chip implementation.

The standard cell binder used for this experiment was a small, automatic package which accessed a local data base of Sandia cells. This package only performed the essential transformations on the graph-expansion of nodes to match the standard cells. However, this package also gives us a worst-case measure for module binding performance.

The translation of the design from one environment, the module binder output, to another, the simulator input, involves both the expansion of multi-bit paths produced by the module binder to the single bit connection format of the simulator input and also the explicit identification of fan-out points. In addition to this latter process, termed resolving, the translator must generate gate specific parameters, such as propagation delays, by compiling capacitances to obtain a realistic simulation using SALOGS. Delay parameters are inserted in the

simulation model by the use of delay gates with associated times.

The input to SALOGS is a description of the network, written in NDL [3]. NDL describes the interconnection of functional blocks. Input and control signals are generated through the SALOGS simulation language SALSIM [4]. In SALOGS, gate representation includes built-in simulator elements such as inverters, transmission gates, NAND, AND, OR, and NOR gates. In addition, any set of elements can be defined as a functional block and the block used as a new element.

The NDL can then be used to automatically generate an IC mask and to determine chip area. The portion of the total data-path area taken up by the different modules is summarized in Figure 7.

Component	Area	% of Total Area	Number of eGates	% of Total Gates	% of Gate subtotal
PC	5124.3	6.9	170	9.4	13.0
LRC	4377.1	5.9	117	6.5	8.9
ADD	3819.5	5.2	117	6.5	8.9
OR	432.0	0.6	18.0	1.0	1.4
AND	476.3	0.6	18.9	1.0	1.4
[illegible]	1411.3	1.9	49	2.7	3.8
INCR	3044.6	4.1	108	6.0	8.3
[illegible]	4217.8	5.7	80	3.3	6.6
12MUX4	3981.8	5.3	56	3.1	4.3
12MUX4	3981.8	5.3	56	3.1	4.3
12MUX4	3981.8	5.3	56	3.1	4.3
12MUX4	3981.8	5.3	56	3.1	4.3
13MUX8	9183.4	12.4	139	7.7	10.8
CADD	1321.8	1.8	45	2.5	3.4
INCR	2644.6	4.8	108	6.0	8.3
LOGIC	1323.8	1.8	45	2.5	3.4
NOR	1323.8	1.8	45	2.5	3.4
subtotal	57867.5	77.2%	1300	72.5%	100.0%
8-NAND2	317.5	0.4	12	0.7	
[illegible]	2981.8	3.3	74	4.1	
2x12MUX2	2044.2	3.8	50	2.8	
1x1MUX2	141.8	0.2	3	0.2	
2-FDL2	158.4	0.2	5	0.3	
SWITCH	1323.8	1.8	45	2.5	
1	1323.8	1.8	45	2.5	
2-FDL12	2118.6	3.0	84	3.5	
LSS	2468.9	3.3	84	4.7	
GEQ	2468.9	3.3	84	4.7	
NEQ	1103.8	1.5	32	1.7	
3-FLOP	338.8	0.5	11	0.6	
TOTAL	74862.8	100.1%	1817	100.0%	

* Gate count is in terms of 2 input NAND gates
** 12MUX4 is 1 of 4 MUX with bit width of 12
*** mc6809 indicates a replica of 8 input EOL comparator

Figure 7: Module Data from Translator

The upper portion of the Figure compares the size of the data-path elements listed in Figure 3. The lower part of the table describes some modules that are more accurately defined as control than data-path. As expected, the percent of sub-total gate count in the upper portion of the table closely resembles the results from the TTL binding shown in Figure 6.

In sum, the CMU PDP-8 design required 74,042 mil-sqr, ignoring the area taken up by routing. The experience with Sandia's IC mask design system indicates that routing takes up about an additional 75% of the area occupied by the standard cells, yielding a chip area of 129,574 mil-sqr. By way of comparison, Intersil's one chip CMOS CPU implementation of the PDP-8 takes up 79,014 mil sqr [6]. It is estimated that 35% of Intersil's CPU chip is devoted to the functional elements equivalent to that generated by the CMU-DA system. Thus, there is a factor of 1.3 difference in the area required for the two designs.

A model can be devised to attribute this seemingly large difference to various parts of the design system. Since each part of the design system builds on top of the previous stage, a multiplicative model is used. This model must take into account the non-optimality of the allocator, non-optimality of the module binder, differences in basic feature size, and the differences in routing techniques. The result of the allocator section indicates a design requiring 1.3 times the size of the DEC design. There is also a difference in the basic feature size of the Intersil and Sandia technologies. By way of comparison, a 12 bit register implemented with Sandia's standard cells occupies 4 times the area of equivalent register in Intersil's design [2]. The multiplicative model then becomes

$$13.8 = (1.3)(4)R,$$

where R is a factor indicating a difference in size between the CMU design and the Intersil design introduced by the module binder. In this case R = 2.5. Not included in the model are factors due to difference between hand packed and channel routing techniques, nor factors considering that large structures (e.g. wide multiplexors) can be more optimally designed by hand than by combination of simple standard cells.

In the worst case, assuming similar input structures and feature size, the CMU module binder would produce a design taking 2.5 times the area of the Intersil design. However, as discussed above, there are other factors which increase the CMU design size that were not accounted for in the model.

9. Summary and Conclusions

The paper has illustrated the methodology behind the CMU Design Automation System. In particular, the datapath of a non-trivial digital system (PDP-8/E) has been designed from an ISPL functional description. Two types of physical modules were bound to the datapath design.

The binding using TTL series modules indicated that the CMU design required 30% more modules than the DEC implementation. The binding using CMOS standard cells indicated that the CMU design is at most a factor of 2.5 off, and due to differences in routing techniques may be actually closer in area to the Intersil design.

As a whole the system has demonstrated the synthesis function in digital system design. The allocator research indicates automated logic synthesis with optimization is feasible and specific module-set information is not necessary in order to produce a reasonable design. The module binding section has demonstrated how the system can design relative to new technologies. Future work with the design system will deal with optimization techniques to be used in better directing the design algorithms for more complex designs.

References

1. Barbacci,M., Barnes,G., Cattell,R., Siewiorek,D. The Symbolic Manipulation of Computer Descriptions ; The ISPS Computer Description Language. Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., March, 1978.

2. Bell, G., C. Mudge, J.E. McNamara. *Computer Engineering*. Digital Press, 1978.

3. Case, G.R. and J.D. Slauter. SALOGS-IV, Program to Perform Logic Simulation and Fault Diagnosis. Proceedings 15th Design Automation Conference, IEEE, 1978.

RCL000205

4. Case, G.R. and J.D. Stauffer. SALSIM - A Langu[illegible]or Control of Digital Logic Simulation. Proceedings of the 11th Annual Asilomar Conference on Circuits, Systems, and Computers, IEEE Circuits and Systems Soc., IEEE Control Systems Soc., Naval Postgraduate School, Univ. of Santa Clara, November, 1977, pp. 370-373.

5. DEC Staff. *PDP-8/E Maintenance Manual.* Digital Equipment Corporation, 1972. DEC-8E-HR1B-D

6. Electronics Magazine Staff. CMOS, Moving Along. *Electronics 48*, 10 (May 1975), .

7. Hafer, L. Data-Memory Allocation in the Distributed Logic Design Style. Master Th., Carnegie-Mellon University, December 1977.

8. Hafer, L.J. and A.C. Parker. Register-Transfer Level Automatic Digital Design: The Allocation Process. Proceedings of the 15th Design Automation Conference, IEEE, 1978.

9. Leive, G.W. The Binding of Modules to Abstract Digital Hardware Descriptions. PhD Thesis Proposal, Carnegie-Mellon University, 1977.

10. Preas, B.T. and C.W. Gwyn. [illegible]llecture For Contemporary Computer Aids to G[illegible]nerate IC Mask Layouts. Proceedings of the 11th Annual Asilomar Conference on Circuits, Systems, and Computers, IEEE Circuits and Systems Soc., IEEE Control Systems Soc., Naval Postgraduate School, Univ. of Santa Clara, November, 1977, pp. 309-317.

11. Sandia Staff. *Standard Cell User's Guide.* Sandia Laboratories, 1978.

12. Snow, E.A., D.P. Siewiorek and D.E. Thomas. A Technology-Relative Computer-Aided Design System: Abstract Representations, Transformations and Design Tradeoffs. Proceedings of the 15th Design Automation Conference, IEEE, June, 1978.

13. Thomas, D.E. and D.P. Siewiorek. Measuring Designer Performance to Verify Design Automated Systems. Proceedings of the 14th Design Automation Conference, IEEE, 1977, pp. 411-418

EXHIBIT 5
(Part 4 of 4)

MAIL ROOM APR 20 1989 PAT. & TRADEMARK OFF

Gp234

PATENT 6/A

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

MJ 4/26/89

H. Kobayashi and M. Shindo
Serial No. 143,821
Filed: January 13, 1988
For: *Knowledge Based Method and Apparatus For Designing Integrated Circuits Using Functional Specifications*

Group Art Unit 234
Examiner: V. Trans

April 18, 1989

Honorable Commissioner of Patents
and Trademarks
Washington, DC 20231

RECEIVED 1989 GROUP 230

AMENDMENT

Sir:

This Amendment is responsive to the rejection mailed January 18, 1989. The Claims are amended to more clearly distinguish them over the prior art. The Specification, the Summary of the Invention and the Abstract of the Disclosure are likewise amended to clarify the distinction. The objections to the drawings are acknowledged and corrections thereto are forthcoming.

In The Specification:

Page 3 line 6, prior to "functional" please add -- architecture independent --.

Page 3 line 14, prior to "functional" please add -- architecture independent --.

Page 3 line 21, prior to "functional" please add -- architecture independent --.

Page 3 line 21, following "into" please delete "a".

Page 3 line 22, prior to "structural" please add -- an architecture specific --.

Page 6 line 10, prior to "representation" please add -- architecture independent --.

Page 6 line 14, following "rectangle" please add -- or box --.

Page 6 line 19, prior to "integrated" please add -- architecture specific --.

Page 7 line 8, following "at" please delete "a".

Page 7 line 9, prior to "behavioral" please add -- an architecture independent functional (--.

Page 7 line 9, following "behavioral" please add --) --.

In The Claims:

Please amend the Claims as follows:

1. (Amended) A computer-aided design system for designing an application specific integrated circuit directly from architecture independent functional specifications for the integrated circuit, comprising

input means operable by a user for defining architecture independent functional specifications for the integrated circuit, and

computer operated means for translating the functional specifications into [a] an architecture specific structural level definition of an integrated circuit.

2. (Amended) The system as defined in Claim 1 wherein said architecture independent functional

specifications are comprised of a series of actions and conditions and wherein said structural level definition [are] is comprised of architecture specific blocks and interconnections between blocks.

5. (Amended) A computer-aided design system for designing an application specific integrated circuit directly from architecture independent functional specifications for the integrated circuit, comprising

a macro library defining a set of [possible] architecture independent operations comprised of actions and conditions;

input specification means operable by a user for defining architecture independent functional specifications for the integrated circuit, said functional specifications being comprised of a series of operations comprised of actions and conditions, said input specification means including means to permit the user to specify for each [action or condition in the defined series of actions and conditions] operation a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing the available operations defined in said macro library [actions and conditions]; and

cell selection means for selecting from said cell library for each macro specified by said input specification means, appropriate hardware cells for performing the operation [action or condition] defined by the specified macro.

Claim 7 line 2, prior to "means" please add -- specification --.

Claim 9 line 2, prior to "means" please add -- specification --.

15. (Amended) A computer-aided design system for designing an application specific integrated circuit directly from a flowchart defining [the] architecture independent functional requirements of the integrated circuit, comprising

a macro library defining a set of [possible] architecture independent operations comprised of actions and conditions;

flowchart editor means operable by a user for creating a flowchart having elements representing said architecture independent operations [actions and conditions];

said flowchart editor means including macro specification means for permitting the user to specify for each [action or condition] operation represented in the flowchart a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing the available operations defined in said macro library [actions and conditions];

cell selection means for selecting from said cell library for each specified macro, appropriate hardware

cells for performing the operation [action or condition] defined by the specified macro; and

data path generator means cooperating with said cell selection means for generating data paths for the hardware cells selected by said cell selector means.

11. (Amended) A computer-aided design system for designing an application specific integrated circuit directly from a flowchart defining [the] architecture independent functional requirements of the integrated circuit, comprising

flowchart editor means operable by a user for creating a flowchart having boxes representing architecture independent actions, diamonds representing architecture independent conditions, and lines with arrows representing transitions between actions and conditions and including means for specifying for each box or diamond, a particular action or condition to be performed;

a cell library defining a set of available integrated circuit hardware cells for performing actions and conditions;

a knowledge base containing rules for selecting hardware cells from said cell library and for generating data and control paths for hardware cells; and

expert system means operable with said knowledge base for translating the flowchart defined by said flowchart editor means into a netlist defining the necessary hardware cells and data and control paths required in an integrated circuit having the specified functional requirements.

20. (Amended) A computer-aided design process for designing an application specific integrated circuit which will perform a desired function comprising

storing a set of definitions of [possible] architecture independent actions and conditions;

storing data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set;

describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions;

specifying for each described action and condition of the series one of said stored definitions which corresponds to the desired action or condition to be performed; and

selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit.

27. (Amended) A knowledge based design process for designing an application specific integrated circuit which will perform a desired function comprising

storing in a macro library a set of macros defining [possible] architecture independent actions and conditions;

storing in a cell library a set of available integrated circuit hardware cells for performing the actions and conditions;

storing in a knowledge base a set of rules for selecting hardware cells from said cell library to perform the actions and conditions defined by the stored macros;

describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions which carry out the function to be performed by the integrated circuit;

specifying for each [describing] described action and condition of said series a macro selected from the macro library which corresponds to the action or condition; and

applying rules of said knowledge base to the specified macros to select from said cell library the hardware cells required for performing the desired function of the application specific integrated circuit.

Claim 28 line 3, preceding "actions" please add -- architecture independent --.

In The Abstract Of The Disclosure:

Please amend the Abstract of the Disclosure as follows:

Page 38 line 4, prior to "functional" please add -- architecture independent --.

Page 38 line 6, prior to "functional" please add -- architecture independent --.

Page 38 line 8, prior to "functional" please add -- architecture independent --.

Page 38 line 9, prior to "functional" please add -- architecture independent --.

Remarks

The present invention is a computer-aided design system and method whereby the user can design application specific integrated circuits at an architecture independent functional behavioral level. By designing at this specification level, the user need not possess the specialized expert knowledge of a highly skilled VLSI design engineer. The architecture independent functional specification of the desired application specific integrated circuit is preferably defined in a flowchart format. The functional level specification is translated by the system into an architecture specific structural level definition which in turn can be used directly to produce the application specific integrated circuit. The definition at the structural level includes a list of integrated circuit hardware cells selected from a cell library needed to achieve the functional specifications. Data paths among the selected hardware cells are also generated by the system. In addition, the system generates a controller and control paths for the selected cells. The preferred embodiment of the system and method for accomplishing translation from an architecture independent functional specification level to a hardware dependent physical layout level is accomplished by a knowledge base utilizing artificial intelligence and expert systems technology. By so doing, the system synthesizes a design logic as it is being translated.

The Examiner has rejected Claims 1-30 under 35 U.S.C. § 103 as being unpatentable over Darringer et al. (U.S. Patent No. 4,703,435). It is respectfully submitted that based upon the Darringer reference, the present invention would not be obvious to one skilled in the art. Although Darringer et al. does disclose a method and system for automatic logic design and it is known in the art of automatic layout to utilize cell libraries of circuit components, Darringer does not teach the present invention. A very clear distinction between Darringer and the present invention is that the input to the Darringer system is in the form of a register transfer level flowchart control language. Darringer et al., U.S. Patent No. 4,703,435, column 4, lines 26-32. In order for a designer to utilize the Darringer system, he/she must possess a sophisticated understanding of the complexities of the circuit logic itself and therefore have the specialized expert knowledge of a highly skilled VLSI design engineer. In contrast, the application specific circuit designer utilizing the present invention need not possess any expertise common among highly skilled VLSI design engineers since input to the present invention is in the form of an architecture independent functional specification.

While Darringer may synthesize logic from a register transfer level flowchart description, it provides no knowledge base of any kind. In contrast, the present invention, as defined in Claims 6, 11, 16, 18, 21, 23, 27 and 29 for example, provides a knowledge base in the form of a rule based automatic logic synthesis component, i.e. an expert system. Thus, Darringer does not teach the method of

synthesis utilized by the present invention. Furthermore, although it is known in the art of automatic layout to utilize hardware cell libraries, a rule based expert system has not been utilized to accomplish a task of selection of cells from the cell library. This clearly distinguishes the present invention over Darringer et al.

Although Darringer et al. does disclose a type of flowchart editor, it is in the form of a noninteractive, nongraphic editor. In contrast, the present invention, as defined in Claims 3, 5, 7, 8, 9, 15, 18 and 28 for example, consists of a front end interactive graphic interface whereby the user of the editor, i.e. the designer, can interactively add, delete and modify elements representing operations in the form of actions and conditions as well state transitions between these elements. Thus, Darringer et al. also does not teach to one skilled in the art the input device for the present invention.

It follows from the above analysis, that the present invention is clearly patentable over Darringer et al. as a result of the input of a design at an architecture independent functional specification level, a knowledge based synthesis of the design during translation, and the front end interactive graphic interface.

The Examiner, alternatively rejected the present invention as unpatentable over Nash et al., *Front End Graphic Interface To The First Silicon Compiler*, European Conference On Electronic Design Automation (EDA 84), March 26-30, 1984, Conference Publication No. 232, pp. 123-124, in light of Darringer et al. It is respectfully submitted that the present invention is likewise clearly patentable over

this cited combination of references. The designer utilizing the Nash et al. system specifies that the design in a layout level description language that is hardware dependent. Examples of this description language appear at Nash et al., pp. 122-133. As in Darringer et al., the designer must possess some of the specialized expert knowledge of a highly skilled VLSI design engineer. In contrast, a designer utilizing the present invention, defines the design at a architecture independent functional specification level and need not have any of the specialized expert knowledge of a highly skilled VLSI design engineer.

Although the Examiner states that Nash et al. discloses a graphic editor driven by a knowledge base, in reality, this limited utility, if it even can be called such, merely provides on-line help information to flag syntactical errors during the editing phase. In contrast, the present invention, utilizes a knowledge base which consists of a rule based expert system to synthesize logic, i.e. data and control path, from an architecture independent functional flowchart description of the circuit design.

Nash provides a front end graphic interface in the form of an interactive layout editor which allows for operations such as placement of primitives, placement of pads, modification of primitives, and creation of a signal mode. In contrast, the present invention provides an interactive flowchart editor and simulator which permits the user to interactively add, delete and modify operations consisting of actions and conditions as well as state transitions between these operations.

The present invention is clearly patentable over Nash et al. as is evident from the above analysis. Furthermore, the present invention is also clearly patentable over Nash et al. in light of Darringer et al. since neither teach to one skilled in the art an architecture independent functional specification level method for designing integrated circuits or a knowledge base for logical synthesis during translation of the design to a hardware dependent description such as a netlist.

In addition to the Darringer et al. and Nash et al. references, several other references were cited by the Examiner as being relevant to this case. The first such reference is L. Trevillyan, *An Overview Of Logic Synthesis Systems*, 24th ACM/IEEE Design Automation Conference, 1987, pp. 166-172. Trevillyan discusses several logic synthesis systems, one of which is the Flamel system. We have obtained and enclosed herewith a complete copy of the Flamel reference cited by Trevillyan - Trickey, H., *Flamel: A High Level Hardware Compiler*, IEEE Transactions On Computer Aided Design, March 1987, pp. 259-269. Input to the Flamel system is in the form of a behavior specification language defined as a subset of Pascal but is associated with a specified bus architecture where the busses interconnect functional units such as ALUs, adders, registers and I/O pads. Trickey at page 259. As a result, a designer utilizing the Flamel system must possess specialized knowledge of computer architectures. It is not an object of the Flamel system to provide a means for designing application specific integrated circuits where the designer lacks this

architecture sophistication. In contrast, however, this is an object of the present invention.

Three other systems which Trevillyan discusses are the Polaris system, the APLAS system and the LSS system. A user of the Polaris and APLAS systems must possess specialized knowledge of a highly skilled VLSI design engineer relating to computer architecture and hardware since input to the systems is in the form of register transfer level languages. Input to the LSS system is in the form of register transfer level languages and source code level language. Therefore, the user of this system must also possess knowledge of computer architecture and hardware. Furthermore, none of these three systems utilize expert systems to accomplish the task of logic synthesis.

The final system which the Trevillyan reference considers is Socrates. Although Socrates is an expert system, it analyzes a netlist rather than generates netlists. Furthermore, input to the Socrates system is in the form of Boolean equations, single output PLAs, or a netlist at the register transfer level. As a result, a designer utilizing the Socrates system, as in all the other systems considered in the Trevillyan reference, must possess the specialized expert knowledge of computer architecture and hardware. Finally, Trevillyan specifically states at page 171 that even though the "synthesis from algorithms" approach is a goal that everyone would like to reach, it is not really practical yet in a production environment. This further clearly distinguishes all these systems considered by Trevillyan from the present invention and shows that the present invention is clearly patentable over those systems.

The two Friedman et al. references, Friedman, T.D. and Yang, S., *Methods Used In An Automatic Logic Design Generator (Alert)*, IEEE Transactions On Computers, Vol. C-18, No. 7, July 1969, pp. 593-614 and Friedman, T.D. and Yang, S., *Quality Of Designs From An Automatic Logic Generator (Alert)*, 7th Design Automation Conference, 1970, pp. 71-89, were cited by the Examiner. The Alert system disclosed in the Friedman et al. references, however, are clearly distinguishable from the present invention and furthermore, the present invention is not obvious from the teachings of the Friedman et al. references. Input to the Alert system is an architectural description in Iverson where the user must define the instruction set, major registers and data flow paths as well as declare storage, channels, important registers, flip-flops and other physical devices. Furthermore, the user must specify memory size, word length, instruction format and other hardware design features. See Friedman et al., *Methods Used In An Automatic Design Generator*, at page 595; Friedman et al., *Quality Of Designs From An Automatic Logic Generator*, at page 71. It is clear, therefore, that a designer using the Alert system must possess the expert knowledge of a highly skilled VLSI design engineer relating to computer architecture and hardware. In contrast, however, a user designing an application specific integrated circuit with the present invention need not possess such sophistication. Furthermore, the output of the present invention is a netlist whereas the Alert system generates Boolean equations of logic block descriptions.

Darringer, John A. et al., *Experiments In Logic Synthesis*, IEEE, 1980, pp. 234-237a cited by the Examiner in turn cites Parker et al., *The CMU Design Automation System - An Example Of Automated Data Path Design*, Proceedings Of The 16th Design Automation Conference, Las Vegas, Nevada, 1979, pp. 73-80. We have obtained Parker et al. and enclose a complete copy herewith. Input to the CMU design automation system is in the form of an algorithmic description in the ISP language. The behavioral description in ISP is translated to the first level path graph with interconnection of abstract components and is then mapped to the next level of data path graph with physical modules selected. Output from the system uses input to the Sandia design system. Therefore, the CMU design automation system as well as the Parker et al. reference do not teach the present invention to one skilled in the art.

A final group of references cited by the Examiner are two Watanabe et al. U.S. patents, Watanabe et al., U.S. Patent No. 4,651,284 and Watanabe et al. U.S. Patent No. 4,700,317. Although the Watanabe references utilize a knowledge base containing rules for layout planning specifying the geometric location of objects, Watanabe utilizes a netlist rather than generates a netlist. Furthermore, the Watanabe system does not synthesize logic nor is synthesis even an object of the system. Therefore, the present invention would not be obvious to one skilled in the art based on the teaching of the Watanabe references thus making the present invention clearly patentable over the Watanabe references.

Several other references are cited by the Examiner but are not relevant to the present invention. They will however be considered in the interest of completeness. Two of these references are papers by Darringer et al., more specifically, Darringer, John A. et al., *Experiments In Logic Synthesis*, IEEE, 1980, pp. 234-237a and Darringer, John A. et al., *A New Look At Logic Synthesis*, 17th Design Automation Conference, 1980, pp. 543-548. As was the case in Darringer et al. U.S. Patent No. 4,703,435, both of these papers by Darringer et al. disclose that even though the input may be in the form of a functional specification, the designer must possess specialized knowledge of computer architecture and possibly even the lower level hardware in order to utilize the systems described. The Daniel et al. reference, Daniel, Marvin E. et al., *CAD Systems For IC Design*, IEEE Transactions On Computer Aided Design Of Integrated Circuits And Systems, Vol. CAD-1, No. 1, January 1982, pp. 2-12, also requires the designer to have specialized expert knowledge common among highly skilled VLSI design engineers. The input form for the Cheng reference, Cheng, Edmund K., *Verifying Compiled Silicon*, VLSI Design, October 1984, pp. 1-4, is a description of high level architecture. As evident from the discussion on page 2, however, the designer must possess specialized knowledge of at least computer architecture and possibly the lower level hardware. For these reasons, the present invention would not be obvious to one skilled in the art based upon these references and furthermore, the present invention is patentably distinct over these prior art references.

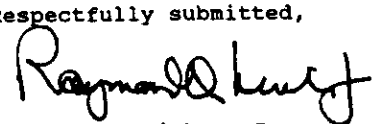
The Examiner also cited several other patents as prior art. Bryant et al., U.S. Patent No. 4,638,442 and Colton et al., Great Britain Patent No. 1,445,914 both require the designer utilizing the systems to possess at least the specialized knowledge of computer architectures and possibly even sophistication of the lower level hardware date logic. Furthermore, Bryant et al. does not provide a knowledge base for synthesis of logic design. For these reasons, the present invention is patentably distinct over Bryant et al. and Colton et al.

Finally, the last two patents cited by the Examiner as prior art were Coleby et al., U.S. Patent No. 4,635,208 and Dunn, U.S. Patent No. 4,656,603. The Coleby reference is concerned with data structures used to link data records containing design information. Although mention of a logical model is made, it is not at the same level as the present invention. Dunn discloses a rule based expert system which utilizes a graphic user interface. However, Dunn does not teach or suggest the specific combination of features of the present invention as defined in the claims.

In view of the amendments and the foregoing remarks, it is submitted that this application is clearly in condition for allowance. Reconsideration by the Examiner and formal notification of the allowance of all claims are respectfully solicited.

RCL000223

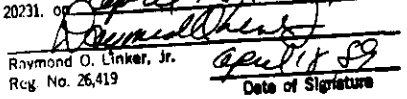
Respectfully submitted,



Raymond O. Linker, Jr.
Registration No. 26,419

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, North Carolina 28234
Telephone: (704) 377-1561
Our File No. 3868-2

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner of Patents and Trademarks, Washington, D.C. 20231. on April 17, 89



Raymond O. Linker, Jr.
Reg. No. 26,419

April 17, 89
Date of Signature

KNG4

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address COMMISSIONER OF PATENTS AND TRADEMARKS
Washington D.C. 20231

SERIAL NUMBER	FILING DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NO.
[illegible]	[illegible]	[illegible] H	3868-2

[illegible]

EXAMINER	
FRANKS W	
ART UNIT	PAPER NUMBER
234	7

DATE MAILED: 08/15/89

This is a communication from the examiner in charge of your application

COMMISSIONER OF PATENTS AND TRADEMARKS

This application has been examined ✓ Responsive to communication filed on April 20, 1989 ✓ This action is made final.

A shortened statutory period for response to this action is set to expire 3 month(s), — days from the date of this letter. Failure to respond within the period for response will cause the application to become abandoned. 35 U.S.C. 133

Part I THE FOLLOWING ATTACHMENT(S) ARE PART OF THIS ACTION:

1. ✓ Notice of References Cited by Examiner, PTO-892.
2. ☐ Notice re Patent Drawing, PTO-948.
3. ✓ Notice of Art Cited by Applicant, PTO-1449
4. ☐ Notice of Informal Patent Application, Form PTO-152
5. ☐ Information on How to Effect Drawing Changes, PTO-1474
6. ☐ _____

Part II SUMMARY OF ACTION

1. ✓ Claims 1 – 30 are pending in the application.

   Of the above, claims _____ are withdrawn from consideration.

2. ☐ Claims _____ have been cancelled.

3. ☐ Claims _____ are allowed.

4. ✓ Claims 1 – 30 are rejected.

5. ☐ Claims _____ are objected to.

6. ☐ Claims _____ are subject to restriction or election requirement.

7. This application has been filed with informal drawings which are acceptable for examination purposes until such time as allowable subject matter is indicated.

8. Allowable subject matter having been indicated, formal drawings are required in response to this Office action.

9. ☐ The corrected or substitute drawings have been received on _____. These drawings are ☐ acceptable; ☐ not acceptable (see explanation).

10. ☐ The ☐ proposed drawing correction and/or the ☐ proposed additional or substitute sheet(s) of drawings, filed on _____ has (have) been ☐ approved by the examiner. ☐ disapproved by the examiner (see explanation).

11. ☐ The proposed drawing correction, filed _____, has been ☐ approved. ☐ disapproved (see explanation). However, the Patent and Trademark Office no longer makes drawing changes. It is now applicant's responsibility to ensure that the drawings are corrected. Corrections MUST be effected in accordance with the instructions set forth on the attached letter "INFORMATION ON HOW TO EFFECT DRAWING CHANGES", PTO-1474.

12. ☐ Acknowledgment is made of the claim for priority under 35 U.S.C. 119. The certified copy has ☐ been received ☐ not been received ☐ been filed in parent application, serial no. _____; filed on _____.

13. ☐ Since this application appears to be in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under Ex parte Quayle, 1935 C.D. 11; 453 O.G. 213.

14. ☐ Other

RCL000224

1. This is responsive to communication filed on April 20, 1989.

2. The following is a quotation of 35 U.S.C. 103 which forms the basis for all obviousness rejections set forth in this Office action:

> "A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made. Subject matter developed by another person, which qualifies as prior art only under subsection (f) and (g) of section 102 of this title, shall not preclude patentability under this section where the subject matter and the claimed invention were, at the time the invention was made, owned by the same person or subject to an obligation of assignment to the same person."

This application currently names joint inventors. In considering patentability of the claims under 35 U.S.C. 103, the examiner presumes that the subject matter of the various claims was commonly owned at the time any inventions covered therein were made absent any evidence to the contrary. Applicant is advised of the obligation under 37 CFR 1.56 to point out the inventor and invention dates of each claim that was not commonly owned at the time a later invention was made in order for the examiner to consider the applicability of potential 35 U.S.C. 102(f) or (g) prior art under 35 U.S.C. 103.

3. Claims 1-30 are rejected under 35 U.S.C. 103 as being unpatentable over Darringer et al (U.S. Patent No. 4,703,435) or Darringer et al in view of Nash et al (EDA84 - A Front End Graphic Interface to the First Silicon Compiler).

Darringer et al disclose a method and system for an automatic logic design that logic is synthesized from a flow-chart level description. It is clear to one skilled in art that a series of action and condition blocks interconnected to form a flow-chart representing a functional specification for the integrated circuit (logic circuit), and it is known in the art of automatic layout that the automatic layout method and system have classically relied on a library of circuit components or cells in the forms of mask geometric defining logic gates to place and interconnet the

cells on a substracte to form an integrated circuit.

As for claims 2 and 15, although Darringer et al do not [illegible] disclose a flow-chart editor means for creating a flow-chart having elements representing actions and conditions [illegible], Nash et al disclose a graphic editor driven by the knowledge base to allow user to creat a schematic or flow-chart on the graphic screen and said schematic or flow-chart to be compiled by LANGUAGE COMPILER. Since both Nash et al and Darringer et al are directed to a system and method for compiling an flow-chart level description through language compiler to generat a physical layout circuit, it would have been obvious to one skilled in the art to employ Nash teaching into Darringer's invention to produce the claimed invention.

4. Applicant's arguments filed April 20, 1999 have been fully considered but are not persuasive.

5. THIS ACTION IS MADE FINAL. Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a). The practice of automatically extending the shortened statutory period an additional month upon the filing of a timely first response to a final rejection has been discontinued by the Office. See 1021 TMOG 35.

A SHORTENED STATUTORY PERIOD FOR RESPONSE TO THIS FINAL ACTION IS SET TO EXPIRE THREE MONTHS FROM THE DATE OF THIS ACTION. IN THE EVENT A FIRST RESPONSE IS FILED WITHIN TWO MONTHS OF THE MAILING DATE OF THIS FINAL ACTION AND THE ADVISORY ACTION IS NOT MAILED UNTIL AFTER THE END OF THE THREE-MONTH SHORTENED STATUTORY PERIOD, THEN THE SHORTENED STATUTORY PERIOD WILL EXPIRE ON THE DATE THE ADVISORY ACTION IS MAILED, AND ANY EXTENSION FEE PURSUANT TO 37 CFR 1.136(a) WILL BE CALCULATED FROM THE MAILING DATE OF THE ADVISORY ACTION. IN NO EVENT WILL THE STATUTORY PERIOD FOR RESPONSE EXPIRE LATER THAN SIX MONTHS FROM THE DATE OF THIS FINAL ACTION.

6. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Vincent Trans whose telephone number is (703) 557 - 8005.

Any inquiry of a general nature or relating to the status of this application should be directed to the Group receptionist whose telephone number is (703) 557-2378.

V. Trans
6/30/89

PARSHOTAM S. LALL
SUPERVISORY PATENT EXAMINER
ART UNIT 234

TO SEPARATE, HOLD TOP AND BOTTOM EDGES, SNAP-APART AND DISCARD CARBON

N.N.

FORM PTO-892 (REV 3-78) U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

NOTICE OF REFERENCES CITED

SERIAL NO.	GROUP ART UNIT	ATTACHMENT TO PAPER NUMBER
07/143,821	234	7

APPLICANT(S): Kobayashi et al.

U.S. PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	NAME	CLASS	SUB-CLASS	FILING DATE IF APPROPRIATE
	A	4803636	2/7/89	Nishiyama et al.	364	491	9/25/86
	B						
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						
	K						

FOREIGN PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	COUNTRY	NAME	CLASS	SUB-CLASS	PERTINENT SHTS. DWG	PERTINENT PP. SPEC.
	L								
	M								
	N								
	O								
	P								
	Q								

OTHER REFERENCES (Including Author, Title, Date, Pertinent Pages, Etc.)

R	
S	
T	
U	

EXAMINER: V. TRANS | DATE: 6/29/89

RCL000227

* A copy of this reference is not being furnished with this office action.
(See Manual of Patent Examining Procedure, section 707.05 (a).)

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

SERIAL NUMBER	FILING DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NO.
07/143,821			

EXAMINER	
TRAN	
ART UNIT	PAPER NUMBER
234	8

DATE MAILED:

EXAMINER INTERVIEW SUMMARY RECORD

All participants (applicant, applicant's representative, PTO personnel):

(1) Ms. R. Linker (3) _____

(2) Eric V. Tran (4) _____

Date of interview Oct 19, 1989

Type: ☐ Telephonic ☑ Personal (copy is given to ☐ applicant ☑ applicant's representative).

Exhibit shown or demonstration conducted: ☐ Yes ☑ No. If yes, brief description: _____

Agreement ☑ was reached with respect to some or all of the claims in question. ☐ was not reached.

Claims discussed: claims 1, 5, 15, 18, 20 and 27

Identification of prior art discussed: Darringer et al. (U.S. Patent 4,703,435)

Description of the general nature of what was agreed to if an agreement was reached, or any other comments: It is agreed that the features "flowchart editor" and "expert system for translating the flowchart into a netlist defining the necessary hardware cells of the integrated circuit" are patentable distinct from the reference identified above. Also, applicant's attorney will amend the claims to include these features.

(A fuller description, if necessary, and a copy of the amendments, if available, which the examiner agreed would render the claims allowable must be attached. Also, where no copy of the amendments which would render the claims allowable is available, a summary thereof must be attached.)

Unless the paragraphs below have been checked to indicate to the contrary, A FORMAL WRITTEN RESPONSE TO THE LAST OFFICE ACTION IS NOT WAIVED AND MUST INCLUDE THE SUBSTANCE OF THE INTERVIEW (e.g., items 1–7 on the reverse side of this form). If a response to the last Office action has already been filed, then applicant is given one month from this interview date to provide a statement of the substance of the interview.

☐ It is not necessary for applicant to provide a separate record of the substance of the interview.

☐ Since the examiner's interview summary above (including any attachments) reflects a complete response to each of the objections, rejections and requirements that may be present in the last Office action, and since the claims are now allowable, this completed form is considered to fulfill the response requirements of the last Office action.

Examiner's Signature

PTOL-413 (REV 1-84)

ORIGINAL FOR INSERTION IN FOLLOW-UP FILE WRAPPER

RCL000228

PATENT

**RESPONSE UNDER 37 CFR 1.116-
EXPEDITED PROCEDURE**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

H. Kobayashi and M. Shindo
Serial No. 07/143,821
Filed: January 13, 1988
For: *Knowledge Based Method and Apparatus For Designing Integrated Circuits Using Functional Specifications*

Group Art Unit 234
Examiner: V. Trans

Honorable Commissioner of Patents
and Trademarks
Box AF
Washington, DC 20231

November 15, 1989

AMENDMENT UNDER 37 C.F.R. 1.116

Sir:

In response to the Official Action mailed August 15, 1989, please amend as follows:

In The Claims:

Please cancel Claims 1-4.

~~5.~~ (Twice Amended) A computer-aided design system for designing an application specific integrated circuit directly from architecture independent functional specifications for the integrated circuit, comprising

a macro library defining a set of architecture independent operations comprised of actions and conditions;

input specification means operable by a user for defining architecture independent functional specifications for the integrated circuit, said functional specifications being comprised of a series of operations comprised of actions and conditions, said input specification means including means to permit the user to

specify for each operation a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing the available operations defined in said macro library; [and]

cell selection means for selecting from said cell library for each macro specified by said input specification means, appropriate hardware cells for performing the operation defined by the specified macro, said cell selection means comprising an expert system including a knowledge base containing rules for selecting hardware cells from said cell library and inference engine means for selecting appropriate hardware cells from said cell library in accordance with the rules of said knowledge base; and

netlist generator means cooperating with said cell selection means for generating as output from the system a netlist defining the hardware cells which are needed to achieve the functional requirements of the integrated circuit and the connections therebetween.

Please cancel Claims 6 and 13.

Claim 14, line 1, please change "13" to --5--.

~~15.~~ (Twice Amended) A computer-aided design system for designing an application specific integrated circuit directly from a flowchart defining architecture independent functional requirements of the integrated circuit, comprising

a macro library defining a set of architecture independent operations comprised of actions and conditions;

flowchart editor means operable by a user for creating a flowchart having elements representing said architecture independent operations;

said flowchart editor means including macro specification means for permitting the user to specify for each operation represented in the flowchart a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing the available operations defined in said macro library;

cell selection means for selecting from said cell library for each specified macro, appropriate hardware cells for performing the operation defined by the specified macro, said cell selection means comprising an expert system including a knowledge base containing rules for selecting hardware cells from said cell library and inference engine means for selecting appropriate hardware cells from said cell library in accordance with the rules of said knowledge base; and

data path generator means cooperating with said cell selection means for generating data paths for the hardware cells selected by said cell selector means, said data path generator means comprising a knowledge base containing rules for selecting data paths between hardware cells and inference engine means for selecting data paths between hardware cells selected by said cell selection means in accordance with the rules of said knowledge base and the arguments of the specified macros.

Please cancel Claim 16.

~~20~~. (Twice Amended) A computer-aided design process for designing an application specific integrated circuit which will perform a desired function comprising

storing a set of definitions of architecture independent actions and conditions;

storing data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set;

storing in an expert system knowledge base a set of rules for selecting hardware cells to perform the actions and conditions;

describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions;

specifying for each described action and condition of the series one of said stored definitions which corresponds to the desired action or condition to be performed; and

selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit, said step of selecting a hardware cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert system knowledge base and generating for the selected integrated circuit hardware cells, a netlist defining the

hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.

Please cancel Claims 21 and 25.

Claim 26, line 1, please change "25" to --20--.

27. (Twice Amended) A knowledge based design process for designing an application specific integrated circuit which will perform a desired function comprising

storing in a macro library a set of macros defining architecture independent actions and conditions;

storing in a cell library a set of available integrated circuit hardware cells for performing the actions and conditions;

storing in a knowledge base a set of rules for selecting hardware cells from said cell library to perform the actions and conditions defined by the stored macros;

describing for a proposed application specific integrated circuit a flowchart comprised of elements representing a series of architecture independent actions and conditions which carry out the function to be performed by the integrated circuit;

specifying for each described action and condition of said series a macro selected from the macro library which corresponds to the action or condition; and

applying rules of said knowledge base to the specified macros to select from said cell library the hardware cells required for performing the desired function

of the application specific integrated circuit and generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.

Please cancel Claim 28.

Remarks

This Amendment cancels several claims, thereby significantly reducing the number of claims under consideration. In addition, independent Claims 5, 15, 20 and 27 have been amended to more definitively distinguish applicants' invention over the prior art. The amendments which have been made to these claims involve rewriting the claims to incorporate language previously set forth in dependent claims. As such, no new issues are presented and entry of this Amendment is therefore clearly proper. Favorable reconsideration by the Examiner in light of this Amendment and allowance of all claims as now presented are earnestly solicited.

The recent interview with Examiner Vincent Trans is acknowledged with appreciation. During this interview the independent claims of record were reviewed in detail with the Examiner and it was pointed out to the Examiner how the present invention distinguishes over the prior art. The prior art of record was discussed in detail in the previous response and, for sake of brevity, these previous remarks will not be repeated. Suffice it to say that the present invention distinguishes fundamentally over the prior art by

providing a system and method for designing an application specific integrated circuit at an architecture independent functional behavioral level. Thus, it is not necessary for the user to have the specialized expert knowledge of a highly skilled VLSI design engineer. The invention makes use of artificial intelligence technology, i.e. an expert system, to translate the architecture independent functional level specifications into an architecture specific structural level definition which can be used directly to produce the application specific integrated circuit. It was noted that while the Darringer et al. patent refers at some points in the specification to a so called "functional description", it is clear from a complete reading of the patent specification in context that the specifications used by Darringer et al. are not truly at an architecture independent level, but rather are at a lower level which is indeed hardware architecture dependent and defines the system at a "register-transfer" level description. This is quite clear from the description at column 5 beginning at line 27.

During the interview, the Examiner carefully reconsidered the prior art and applicants' claims, and upon reconsideration agreed that certain features as defined in applicants' claims, such as the "flowchart editor" and the "expert system for translating the flowchart into a netlist defining the necessary hardware cells of the integrated circuit" patentably distinguish applicants' invention from the prior art of record, including Darringer et al. 4,703,435. Thus, it was agreed that Claim 18 in its present form, for example, patentably defines applicants' invention over the prior art of record. Thus, Claim 18 (and Claim 19

which is dependent therefrom) should now be in condition for allowance.

By the present Amendment applicants have cancelled Claims 1-4 and have amended the remaining independent claims so as to place them in condition for allowance. Claim 15 for example has been amended to more clearly recite the use in the system of the present invention of a knowledge base and an inference engine for selecting the hardware cells and for generating data paths between hardware cells. It is submitted that the combination of structural features as recited in Claim 15 defines a novel and unobvious departure from the prior art. Accordingly, Claim 15 and dependent Claim 17 should be in condition for allowance.

Claim 27 is directed to the method aspects of applicants' invention. This claim has been amended to recite the utilization of a flowchart and to additionally recite the generation of a netlist defining the hardware cells of the integrated circuit and the interconnection requirements therefor. This claim and Claims 29 and 30 which are dependent therefrom clearly distinguish applicants' invention over the prior art.

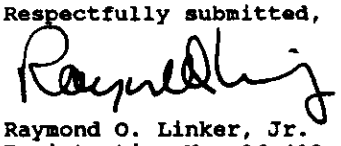
Claim 5 has also been amended to clearly distinguish it over the cited prior art by more clearly defining the expert system aspects of applicants' invention including the provision of a knowledge base containing rules for selecting hardware cells, inference engine means for selecting appropriate hardware cells, and netlist generator means for generating a netlist defining the hardware cells which are needed to perform the functional requirements of the integrated circuit and the connections therebetween. The combination of features as defined in Claim 5 is not

identically shown in any of the cited prior art, and moreover the differences between the invention as defined in this claim and the prior art are quite significant and are not suggested by or obvious from the prior art of record.

Independent Claim 20 has been also amended to emphasize the expert system aspects of applicants' method. Claim 20 as now presented thus defines a combination of method steps which are neither shown nor suggested by the prior art of record.

For the reasons noted it is submitted that all of the claims of the application as now presented are in condition for allowance. Favorable reconsideration by the Examiner, entry of this Amendment, and notification of the allowability of all claims as now presented are earnestly solicited.

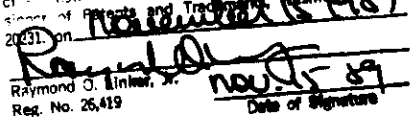
Respectfully submitted,



Raymond O. Linker, Jr.
Registration No. 26,419

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, North Carolina 28234
Telephone: (704) 377-1561
ROLjr:kcs
Our File No. 3868-2

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner of Patents and Trademarks, Washington, D.C. 20231 on



Raymond O. Linker, Jr.
Reg. No. 26,419

Date of Signature

UNITED STATE "PARTMENT OF COMMERCE
Patent and Trac ark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

SERIAL NUMBER	FILING DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NO.

EXAMINER	
ART UNIT	PAPER NUMBER
234	10

DATE MAILED: 11/29/89

NOTICE OF ALLOWABILITY

PART I.

1. ☑ This communication is responsive to the amendment filed on Nov. 20, 1989.
2. ☑ All the claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice Of Allowance And Issue Fee Due or other appropriate communication will be sent in due course.
3. ☑ The allowed claims are 5,7-12,14,15,17-20,22-24,26,27,29 & 30 (Now renumbered 1-20)
4. ☐ The drawings filed on _____ are acceptable.
5. ☐ Acknowledgment is made of the claim for priority under 35 U.S.C. 119. The certified copy has [_] been received. [_] not been received. [_] been filed in parent application Serial No. _____, filed on _____.
6. ☑ Note the ~~attached~~ Examiner's Amendment.
7. ☐ Note the attached Examiner Interview Summary Record, PTOL-413.
8. ☐ Note the attached Examiner's Statement of Reasons for Allowance.
9. ☐ Note the attached NOTICE OF REFERENCES CITED, PTO-892.
10. ☐ Note the attached INFORMATION DISCLOSURE CITATION, PTO-1449.

PART II.

A SHORTENED STATUTORY PERIOD FOR RESPONSE to comply with the requirements noted below is set to EXPIRE THREE MONTHS FROM THE "DATE MAILED" indicated on this form. Failure to timely comply will result in the ABANDONMENT of this application. Extensions of time may be obtained under the provisions of 37 CFR 1.136(a).

1. ☐ Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL APPLICATION, PTO-152, which discloses that the oath or declaration is deficient. A SUBSTITUTE OATH OR DECLARATION IS REQUIRED.
2. ☑ APPLICANT MUST MAKE THE DRAWING CHANGES INDICATED BELOW IN THE MANNER SET FORTH ON THE REVERSE SIDE OF THIS PAPER.
   a. ☑ Drawing informalities are indicated on the NOTICE RE PATENT DRAWINGS, PTO-948, attached hereto or to Paper No. 3. CORRECTION IS REQUIRED.
   b. ☐ The proposed drawing correction filed on _____ has been approved by the examiner. CORRECTION IS REQUIRED.
   c. ☐ Approved drawing corrections are described by the examiner in the attached EXAMINER'S AMENDMENT. CORRECTION IS REQUIRED.
   d. ☑ Formal drawings are now REQUIRED.

Any response to this letter should include in the upper right hand corner, the following information from the NOTICE OF ALLOWANCE AND ISSUE FEE DUE: ISSUE BATCH NUMBER, DATE OF THE NOTICE OF ALLOWANCE, AND SERIAL NUMBER.

Attachments:

- _ Examiner's Amendment
- _ Examiner Interview Summary Record, PTOL-413
- _ Reasons for Allowance
- _ Notice of References Cited, PTO-892
- _ Information Disclosure Citation, PTO-1449
- _ Notice of Informal Application, PTO-152
- _ Notice re Patent Drawings, PTO-948
- _ Listing of Bonded Draftsmen
- _ Other

EXAMINER'S AMENDMENT:

(Note: the followings are of obvious type editorial.)

In claim 22, line 1, change "21" to --20--;

In claim 26, line 1, change "25" to --20--.

RCL000238

V. Trans
(703) 557-4169

Felix D. Gruber

FELIX D. GRUBER
PRIMARY EXAMINER
ART UNIT 234

PTOL-37 (REV. 2-85)

USCOMM-DC 85-3744

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office

Address: Box ISSUE FEE
COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

NOTICE OF ALLOWANCE AND ISSUE FEE DUE

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED.

THE ISSUE FEE MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED.

HOW TO RESPOND TO THIS NOTICE:

I. Review the SMALL ENTITY Status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the Status is changed, pay twice the amount of the FEE DUE shown above and notify the Patent and Trademark Office of the change in status, or

B. If the Status is the same, pay the FEE DUE shown above.

If the SMALL ENTITY is shown as NO:

A. Pay FEE DUE shown above, or

B. File verified statement of Small Entity Status before, or with, payment of 1/2 the FEE DUE shown above.

II. Part B of this notice should be completed and returned to the Patent and Trademark Office (PTO) with your ISSUE FEE. Even if the ISSUE FEE has already been paid by a charge to deposit account, Part B should be completed and returned. If you are charging the ISSUE FEE to your deposit account, Part C of this notice should also be completed and returned.

III. All communications regarding this application must give series code (or filing date), serial number and batch number. Please direct all communications prior to issuance to Box ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees.

RCL000239

PTOL-85 (REV 12-88)(OMB Clearance is pending)

ROOM JAN 4 1990 PATENT & TRADEMARK OFFICE

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Hideaki Kobayashi, et al.
Serial No. 07/143,821
Filed: January 13, 1988
For: KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS

Date of Notice of Allowance: November 29, 1989

Issue Batch No. B02

The Honorable Commissioner of
Patents and Trademarks
Washington, DC 20231

January 2, 1990

SUBMITTAL OF FORMAL DRAWINGS

Sir:

In response to the requirement for new drawings as set forth in Paper No. 3 and the Notice of Allowability mailed November 29, 1989 in the above application, there is enclosed herewith one set (12 sheets) of new formal drawings. It is requested that these new drawings be substituted for the originally filed informal drawings.

Respectfully submitted,

Raymond O. Linker, Jr.
Registration No. 26,419

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, North Carolina 28234
Telephone: (704) 377-1561
ROLjr:kcs
Our File No. 3868-2

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner of Patents and Trademarks, Washington, D.C. 20231 on January 2, 1990

Raymond O. Linker, Jr.
Reg. No. 26,419

January 2, 1990
Date of Signature

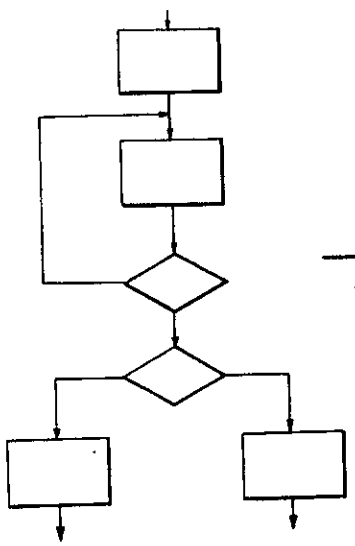


Fig. 1a.
FUNCTIONAL LEVEL

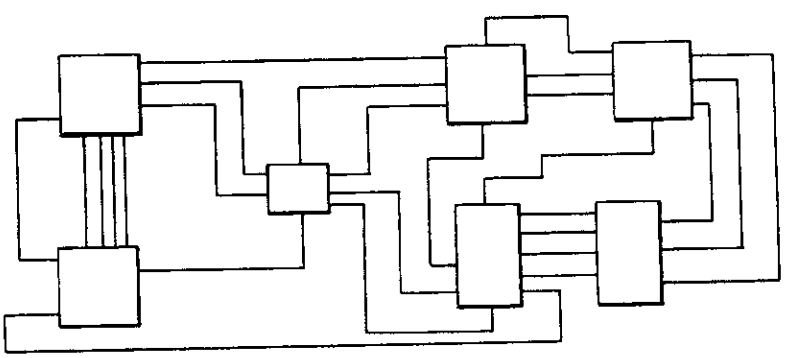


Fig. 1b.
STRUCTURAL LEVEL

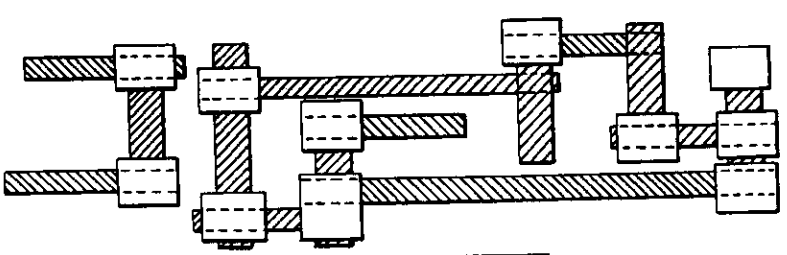


Fig. 1c.
PHYSICAL LAYOUT LEVEL

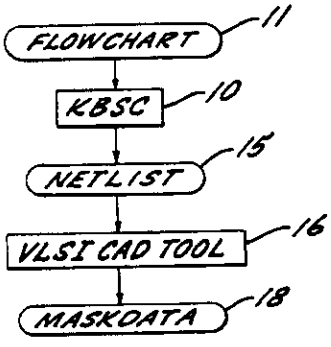

FLOWCHART
11
KBSC
10
NETLIST
15
VLSI CAD TOOL
16
MASKDATA
18

Fig. 2.

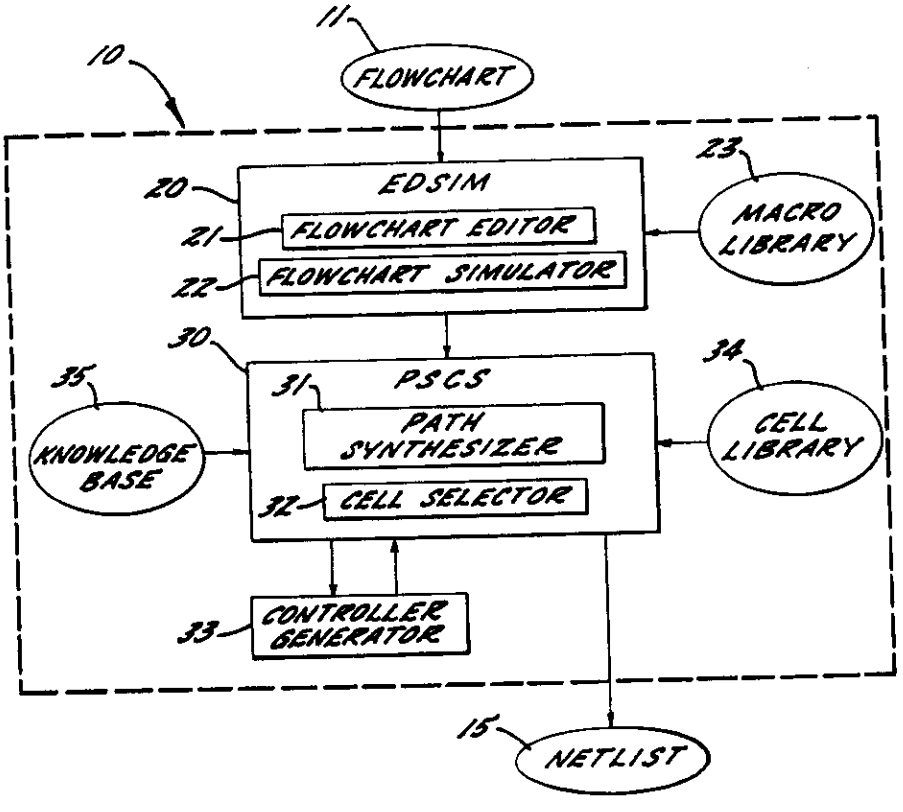

11
FLOWCHART
10
EDSIM
20
FLOWCHART EDITOR
21
FLOWCHART SIMULATOR
22
23
MACRO LIBRARY
30
PSCS
31
PATH SYNTHESIZER
CELL SELECTOR
32
35
KNOWLEDGE BASE
34
CELL LIBRARY
CONTROLLER GENERATOR
33
15
NETLIST

Fig. 3.

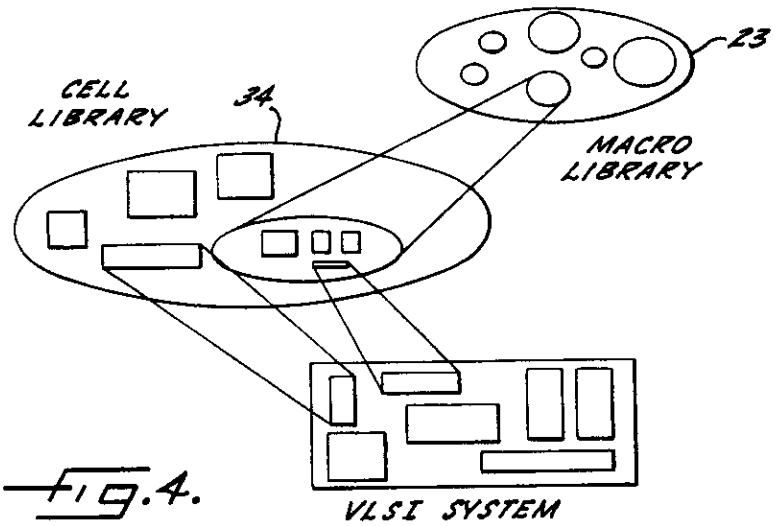

23
CELL LIBRARY
34
MACRO LIBRARY
VLSI SYSTEM

Fig. 4.

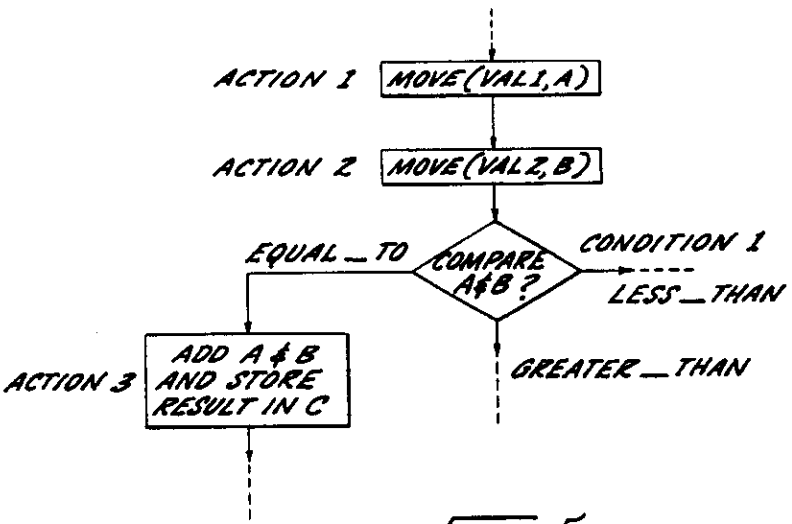

ACTION 1
MOVE(VAL1,A)
ACTION 2
MOVE(VAL2,B)
EQUAL_TO
COMPARE A&B ?
CONDITION 1
LESS_THAN
GREATER_THAN
ACTION 3
ADD A & B AND STORE RESULT IN C

Fig. 5.

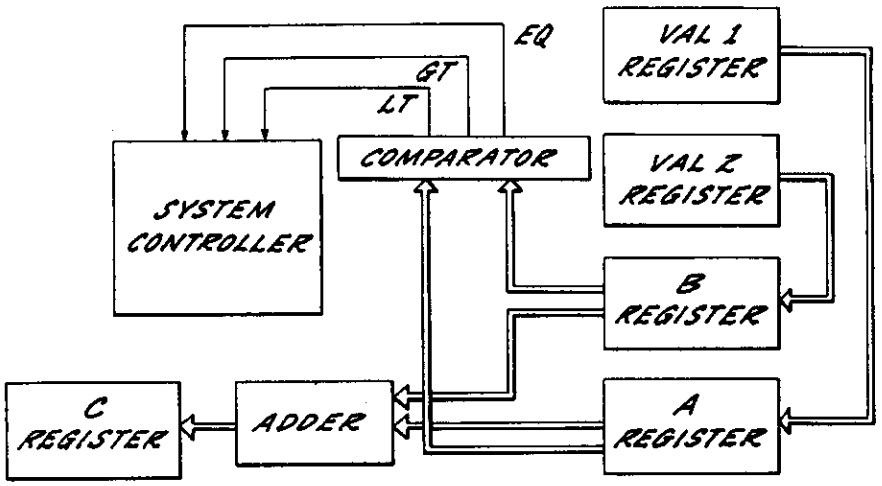

EQ
GT
LT
VAL 1 REGISTER
COMPARATOR
VAL 2 REGISTER
SYSTEM CONTROLLER
B REGISTER
C REGISTER
ADDER
A REGISTER

Fig. 6.

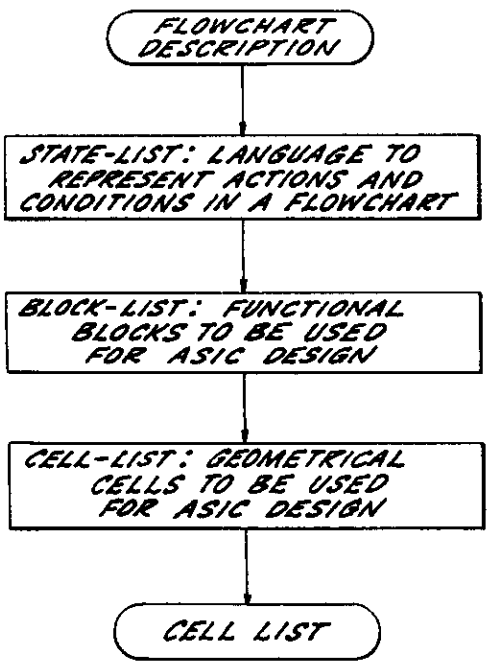

FLOWCHART DESCRIPTION
STATE-LIST: LANGUAGE TO REPRESENT ACTIONS AND CONDITIONS IN A FLOWCHART
BLOCK-LIST: FUNCTIONAL BLOCKS TO BE USED FOR ASIC DESIGN
CELL-LIST: GEOMETRICAL CELLS TO BE USED FOR ASIC DESIGN
CELL LIST

Fig. 9.

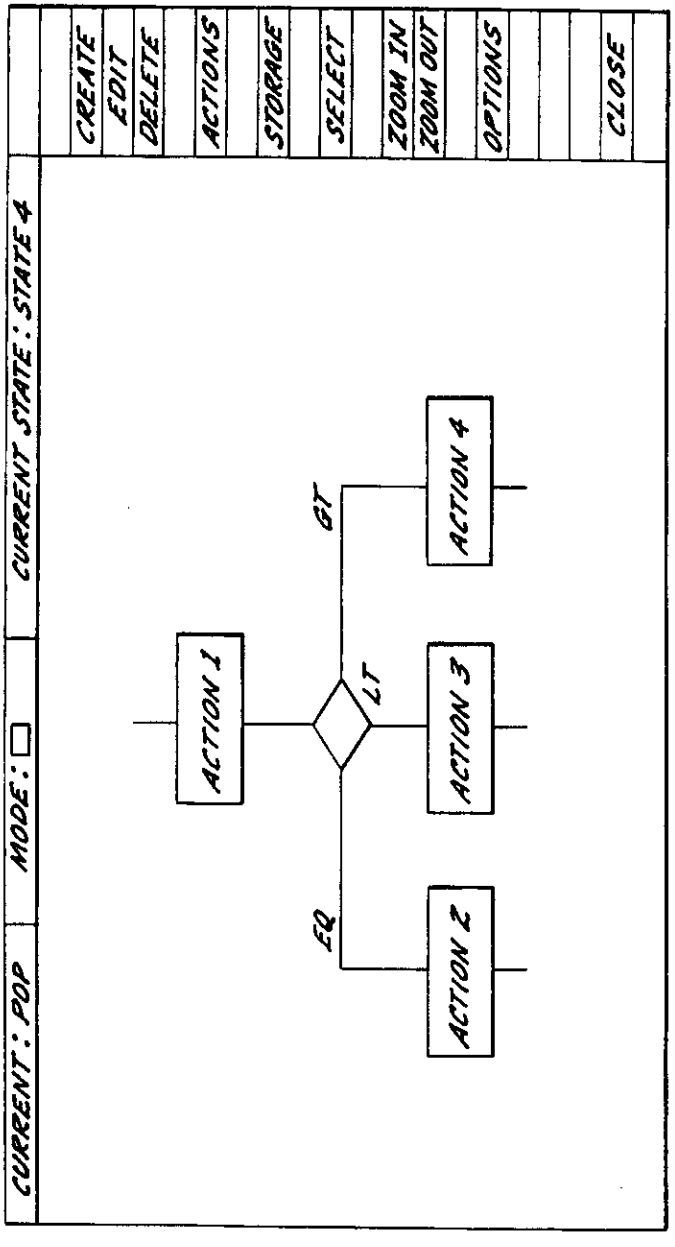

CREATE
EDIT
DELETE
ACTIONS
STORAGE
SELECT
ZOOM IN
ZOOM OUT
OPTIONS
CLOSE
CURRENT STATE : STATE 4
MODE : □
CURRENT : POP
ACTION 1
GT
LT
EQ
ACTION 4
ACTION 3
ACTION 2

FIG. 7.

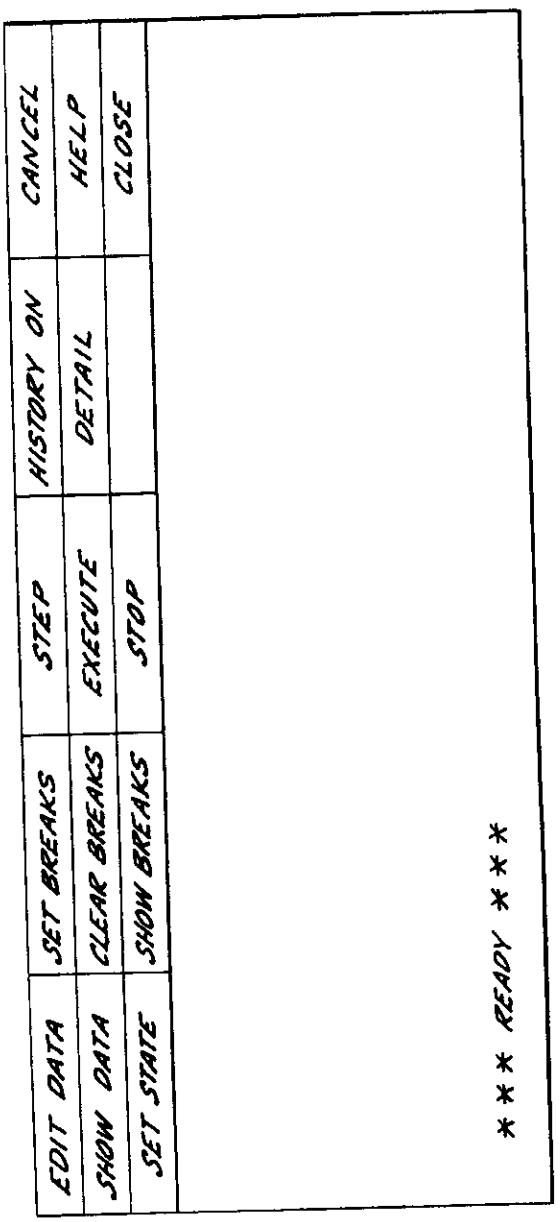

EDIT DATA
SHOW DATA
SET STATE
SET BREAKS
CLEAR BREAKS
SHOW BREAKS
STEP
EXECUTE
STOP
HISTORY ON
DETAIL
CANCEL
HELP
CLOSE
*** READY ***

FIG. 8.

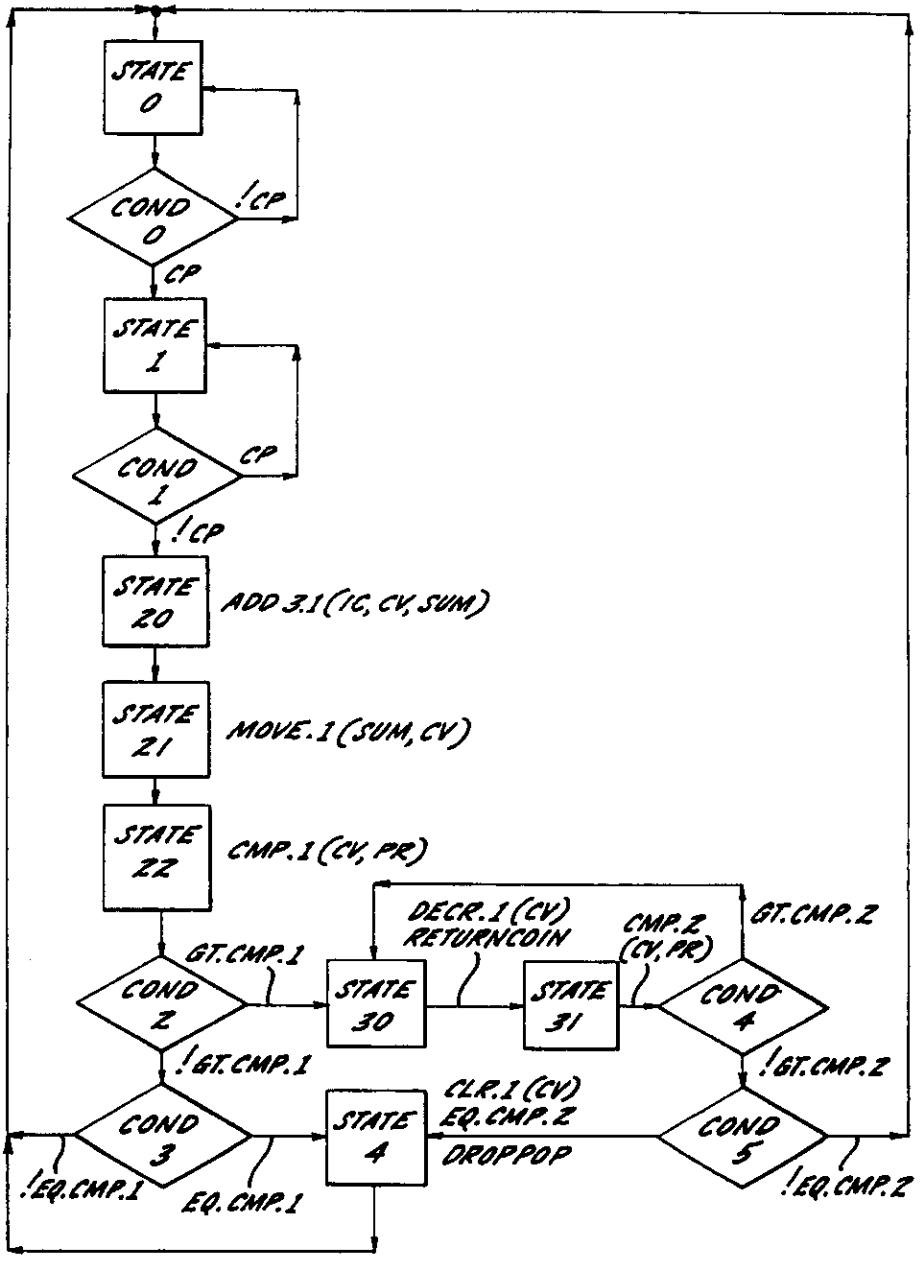


Fig. 10.

RCL000247

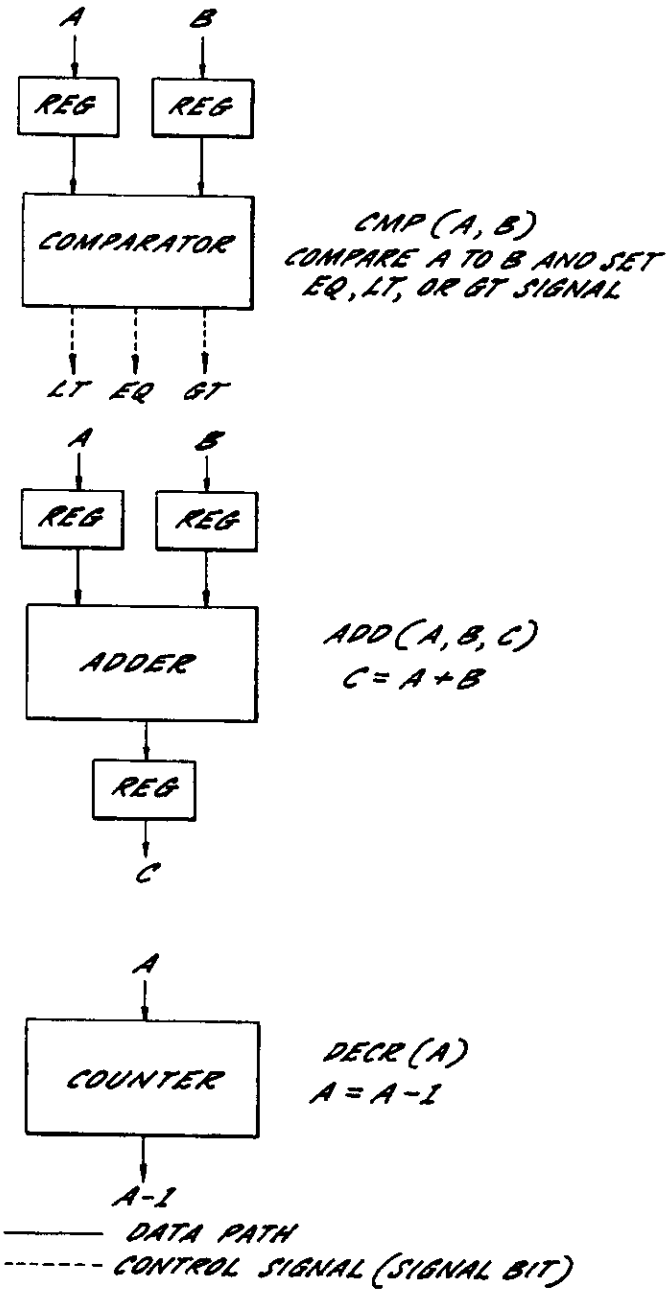

A
B
REG
REG
COMPARATOR
CMP (A, B)
COMPARE A TO B AND SET
EQ, LT, OR GT SIGNAL
LT EQ GT
A
B
REG
REG
ADDER
ADD (A, B, C)
C = A + B
REG
C
A
COUNTER
DECR (A)
A = A - 1
A-1
DATA PATH
CONTROL SIGNAL (SIGNAL BIT)

Fig. 11.

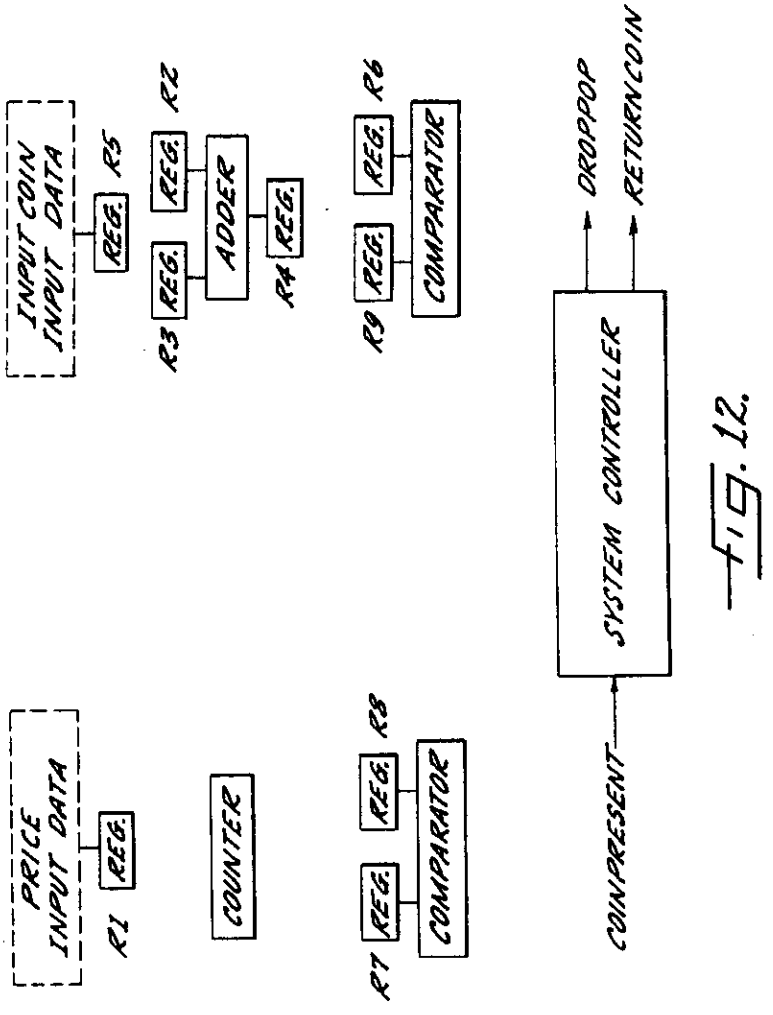

INPUT COIN/ INPUT DATA
REG.
R5
REG.
REG.
R2
R3
ADDER
R4
REG.
REG.
R6
R9
COMPARATOR
DROPPOP
RETURNCOIN
SYSTEM CONTROLLER
COINPRESENT
PRICE INPUT DATA
REG.
R1
COUNTER
REG.
R8
REG.
R7
COMPARATOR

FIG. 12.

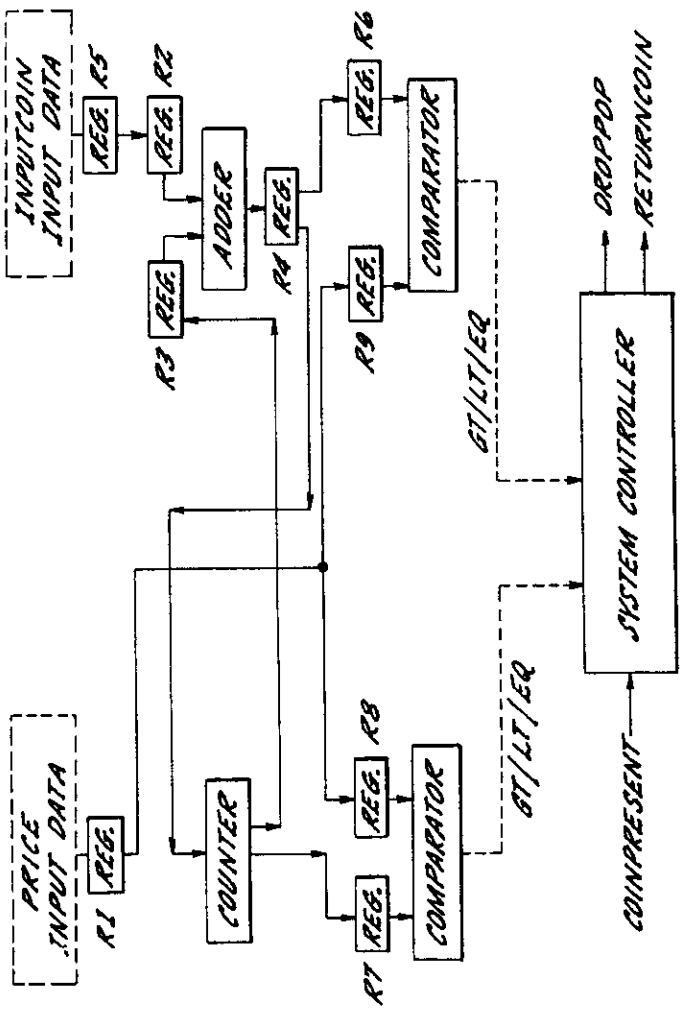

INPUT COIN
INPUT DATA
REG.
R5
R2
ADDER
R4
R6
COMPARATOR
R3
R9
GT/LT/EQ
DROP POP
RETURN COIN
SYSTEM CONTROLLER
PRICE
INPUT DATA
R1
COUNTER
R8
R7
COIN PRESENT

FIG. 13.

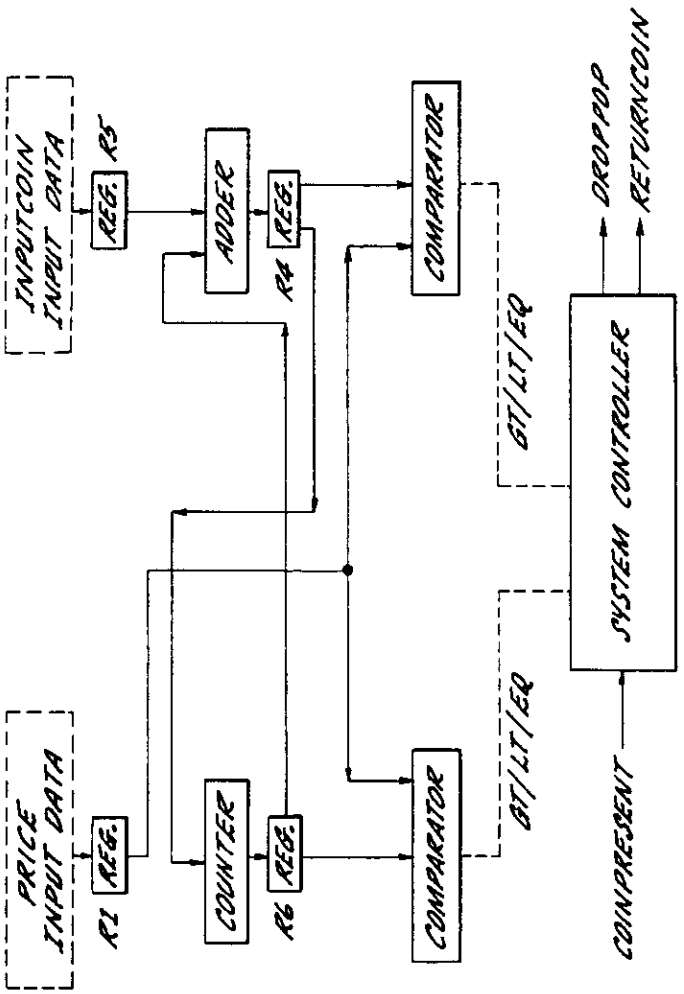

INPUTCOIN
INPUT DATA
REG.
R5
ADDER
REG.
R4
COMPARATOR
DROPPOP
RETURNCOIN
GT/LT/EQ
SYSTEM CONTROLLER
PRICE
INPUT DATA
REG.
R1
COUNTER
REG.
R6
COMPARATOR
GT/LT/EQ
COINPRESENT

FIG. 14.

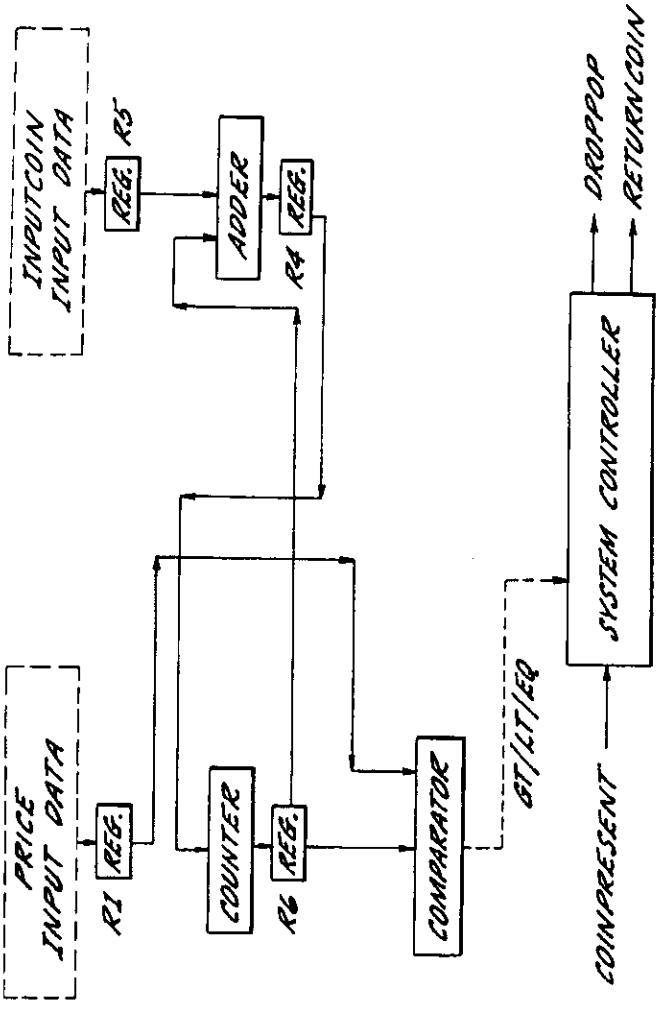

INPUTCOIN
INPUT DATA
REG.
R5
ADDER
REG.
R4
DROPPOP
RETURNCOIN
SYSTEM CONTROLLER
PRICE
INPUT DATA
REG.
R1
COUNTER
REG.
R6
COMPARATOR
GT/LT/EQ
COINPRESENT

FIG. 15.

10.00-142 B

PART B - ISSUE FEE TRANSMITTA.

MAILING INSTRUCTIONS: This form st ...used for transmitting the ISSUE FEE. Blocks 2 thro... completed where appropriate. All further correspondence including the issue Fe... receipt, the Patent, advanced orders and notification of ... ...mainte... fees will be mailed to addressee listed in Block 1, unless you direct otherwise. Ply. (a) specifying a new correspondence address in Block 3 below; or (b) providing the PTO with a separate FEE ADDRESS for maintenance fee notifications with the payment of Issue Fee or thereafter. See reverse for Certificate of Mailing.

1. CORRESPONDENCE ADDRESS

[illegible] Lipsey, Jr.
[illegible]
[illegible]
[illegible] NC 28234

2. INVENTOR(S) ADDRESS CHANGE (Complete only if there is a change)

INVENTOR'S NAME

Street Address

City, State and ZIP Code

CO-INVENTOR'S NAME

Street Address

City, State and ZIP Code

☐ Check if additional changes are on reverse side

4/ECT

APPLICATION NO. | FILING DATE | EXAMINER AND GROUP ART UNIT

07/143,821 01/13/88 020 TRANS, V 234 11/[illegible]/89

INVENTOR(S)

KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS

ATTORNEY DOCKET NO. | CLASS-SUBCLASS | BATCH NO. | APPLN. TYPE | SMALL ENTITY | FEE DUE | DATE DUE

[illegible] 364-490.000 802 UTILITY NO $620.00 02/28/90

3. Further correspondence to be mailed to the following:

Bell, Seltzer, Park & Gibson, P.A.
Post Office Drawer 34009
Charlotte, North Carolina 28234

4. For printing on the patent front page, list the names of not more than 3 registered patent attorneys or agents OR alternatively, the name of a firm having as a member a registered attorney or agent. If no name is listed, no name will be printed.

1 Bell, Seltzer, Park &
2 Gibson, P.A.
3

DO NOT USE THIS SPACE

070 03/01/90 07143821

RCL000253

1 142 620.00 CK

5. ASSIGNMENT DATA TO BE PRINTED ON THE PATENT (print or type)

(1) NAME OF ASSIGNEE:
1. International Chip Corporation
2. Ricoh Company, Ltd.

(2) ADDRESS (City, State or Country)
1. Columbia, South Carolina
2. Tokyo, Japan

(3) STATE OF INCORPORATION, IF ASSIGNEE IS A CORPORATION
1. South Carolina
2. Japan

A. ☐ This application is NOT assigned.

☒ Assignment previously submitted to the Patent and Trademark Office.

☐ Assignment is being submitted under separate cover. Assignments should be directed to Box ASSIGNMENTS.

PLEASE NOTE: Unless an assignee is identified in Block 5, no assignee data will appear on the patent. Inclusion of assignee data is only appropriate when an assignment has been previously submitted to the PTO or is being submitted under separate cover. Completion of this form is NOT a substitute for filing an assignment.

6a. The following fees are enclosed:

☒ Issue Fee ☐ Advanced Order - # of Copies ____ (Minimum of 10)

6b. The following fees should be charged to:

DEPOSIT ACCOUNT NUMBER 16-0605 (Enclose Part C)

☐ Issue Fee ☐ Advanced Order - # of Copies ____ (Minimum of 10)

☒ Any Deficiencies in Enclosed Fees

The COMMISSIONER OF PATENTS AND TRADEMARKS is requested to apply the Issue Fee to the application identified above.

(Signature of party in interest of record)
Raymond O. Linker, Jr.
Reg. No. 29,419

(Date) 2/19/90

NOTE: The Issue Fee will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the Patent and Trademark Office.

[illegible] CERTIFICATE OF MAILING ON REVERSE

[illegible] (OMB Clearance is pending)

Certificate of Mailing

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to:

Box ISSUE FEE
Commissioner of Patents and Trademarks
Washington, D.C. 20231

on February 19, 1990
(Date)

Raymond O. Linker, Jr., Reg. No. 26,419
(Name of person making deposit)

Raymond O. Linker
(Signature)

Feb 19 1990
(Date)

Note: If this certificate of mailing is used, it can only be used to transmit the Issue Fee. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawings, must have its own certificate of mailing.

03/01/90 07143851 070
1 142
CK 00.052

This form is estimated to take 20 minutes to complete. Time will vary depending upon the needs of the individual applicant. Any comments on the amount of time you require to complete this form should be sent to the Office of Management and Organization, Patent and Trademark Office, Washington, D.C. 20231 and to the Office of Information and Regulatory Affairs, Office of Management and Budget, Washington, D.C. 20503.

RCL000254

VERSE PTOL-85B (REV 12-88)(OMB Clearance is pending)

PAUL B. GE...
DONALD M. S...
CHARLES B. PARK, III
FLOYD A. GIBSON
SAMUEL G. LAYTON, JR.
JULIAN E. CARNES, JR.
JOELL T. FLANER
CHARLES B. ELDERKIN
JOHN L. SULLIVAN, JR.
JOHN J. BARNHARDT, III
RAYMOND O. LINKER, JR.
JAMES O. MYERS
MICHAEL D. McCOY
BLAS P. ARROYO
JOSEPH H. HEARD
PHILIP SUMMA
MITCHELL S. BIGEL
KENNETH O. SIBLEY
DICKSON M. LUPO
RICHARD K. WARTHER
F. MICHAEL SAJOVEC
JOHN H. THOMAS
BARBARA K. CALDWELL
MARTHA G. BARBER
FRANK BURKHEAD WYATT, II
PAUL F. PEDIGO
MICHAEL S. CONNOR
GEORGE M. TAULBEE
BRIAN P. O'SHAUGHNESSY
CHRISTOPHER F. REGAN

BELL, SELTZER, PARK & GIBSON
A PROFESSIONAL ASSOCIATION
ATTORNEYS AT LAW
1211 EAST MOREHEAD STREET
P. O. DRAWER 34009
CHARLOTTE, N. C. 28234
704-377-1561
FACSIMILE 704-334-2014

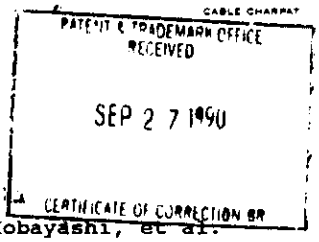
INTELLECTUAL PROPERTY LAW
PATENTS, TRADEMARKS, COPYRIGHTS
& TRADE SECRETS; LICENSING
ANTITRUST; UNFAIR TRADE PRACTICES;
TECHNOLOGY AND COMPUTER LAW

RESEARCH TRIANGLE AREA OFFICE
310 UCB PLAZA
3605 GLENWOOD AVENUE
P. O. DRAWER 31107
RALEIGH, N. C. 27622
919-881-3140
FACSIMILE 919-881-3175

TELEX 57-5102
CABLE CHARPAT

C of C
#13 LW

September 17, 1990



The Honorable Commissioner of Patents and Trademarks
Washington, D.C. 20231

Re: United States Patent of Hideaki Kobayashi, et al.
Serial No. 07/143,821
Filed: January 13, 1988
Patent No. 4,922,432
Issued: May 1, 1990
Our File 3868-2

Sir:

It is respectfully requested that a Certificate of Correction be issued for the above-identified patent, in accordance with Rule 322. This request is made in order to correct the mistakes incurred through the fault of the Patent Office.

The mistakes appearing in the patent are set forth on the Certificate of Correction enclosed herewith, with an additional copy thereof and a postal card being enclosed in accordance with the present Patent Office practice.

Respectfully submitted,

Raymond O. Linker, Jr.
Registration No. 26,419

ROLjr/kcs

Enclosures

Staple Here Only

PRINTER'S TRIM LINE

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. :4,922,432

DATED :May 1, 1990

INVENTOR(S) :Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On the face of the patent under the section "References Cited" under "Other Publications":

"Verifying Compiled Silicon", by E. K. cheng, VLSI Design, Oct. 1984, pp. 1-4." should be -- "Verifying Compiled Silicon", by E. K. Cheng, VLSI Design, Oct. 1984, pp. 1-4." --.

"quality of Designs from An Automatic Logic Generator", by T. D. Friedman et al., IEEE 7th DA Conference, 1970, pp. 71-89." should be -- "Quality of Designs from An Automatic Logic Generator", by T. D. Friedman et al., IEEE 7th DA Conference, 1970, pp. 71-89. --.

"Trevillyan-Trickey, H., Flamel: *A High Level Hardward Compiler*, IEEE Transactions On Computer Aided Design, Mar. 1987, pp. 259-269." should be -- Trevillyan-Trickey, H., Flamel: *A High Level Hardware Compiler*, IEEE Transactions On Computer Aided Design, Mar. 1987, pp. 259-269. --.

In the abstract:

Every occurrence of "functional architecture independent" should be -- architecture independent functional --.

In the Specification:

Column 1, line 19, "a" should be -- an --.

MAILING ADDRESS OF SENDER:
Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, NC 28234

PATENT NO. 4,922,432

No. of add'l. copies @ 30¢ per page

FORM PTO 1050 (REV. 3-82)

RCL000256

Staple Here Only !

PRINTER'S TAIM LINE

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 2, line 10, "functional architecture independent" should be -- architecture independent functional --. clr

Column 2, line 21, "functional architecture independent" should be -- architecture independent functional --. clr

Column 2, lines 29-30, "functional architecture independent" should be -- architecture independent functional --. clr

Column 2, line 31, "structural" should be after "specific". clr

Column 3, lines 51-52, "representation" should be after "architecture independent". clr

Column 3, lines 61-62, "integrated" should be after "specific". clr

Column 6, line 62, after "22" insert -- . --. P

Column 7, line 43 (in Table 1), "C = A B" should be -- C = A^B --. P

Column 8, line 9 should end with the word "flowchart" and "history" should begin on the next line. P

MAILING ADDRESS OF SENDER:
Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, NC 28234

PATENT NO. 4,922,432

No. of add'l. copies @ 30¢ per page

FORM PTO 1050 (REV. 3-82)

RCL000257

Staple there Only !

PRINTER'S TRIM LINE

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 10, line 23, "data paths" should be -- datapaths --.

Column 10, line 68, delete "The rule format to be used is as follows:"

Column 12, line 54, "Enqineering" should be -- Engineering --.

Column 13, line 55, "block" should be -- blocks --.

In the Claims:

Column 14, line 68, before "means" (first occurrence) insert -- specification --; after "means" (second occurrence) delete "specification".

Column 15, line 9, before "means" (first occurrence) insert -- specification --; after "means" (second occurrence) delete "specification".

Column 15, line 35, after "circuit" insert -- , --.

Column 15, line 36, "marco" should be -- macro --.

Column 15, line 49, "form" should be -- from --.

MAILING ADDRESS OF SENDER:

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, NC 28234

PATENT NO. 4,922,432

No. of @ 30

FORM PTO 1050 (REV. 3-82)

RCL000258

Staple Here Only !

PRINTER'S TRIM LINE

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 16, line 14, "condition" should be -- conditions --.

Column 17, line 19, after "base" insert -- a --.

MAILING ADDRESS OF SENDER:

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, NC 28234

PATENT NO. 4,922,432

No. of add'l. copies @ 30¢ per page

FORM PTO 1050 (REV. 3-82)

RCL000259

Staple Here Only !

PRINTER'S TRIM LINE

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 16, line 14, "condition" should be -- conditions --.

Column 17, line 19, after "base" insert -- a --.

MAILING ADDRESS OF SENDER:

Bell, Seltzer, Park & Gibson
Post Office Drawer 34009
Charlotte, NC 28234

PATENT NO. 4,922,432

No. of add'l. copies @ 30¢ per page

RCL000260

FORM PTO 1050 (REV. 3-82)

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432 Page 1 of 4

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

ON TITLE PAGE: under the section "References Cited" under "Other Publications":

"Verifying Compiled Silicon", by E. K. cheng, VLSI Design, Oct. 1984, pp. 1-4." should be -- "Verifying Compiled Silicon", by E. K. Cheng, VLSI Design, Oct. 1984, pp. 1-4." --.

"quality of Designs from An Automatic Logic Generator", by T. D. Friedman et al., IEEE 7th DA Conference, 1970, pp. 71-89." should be -- "Quality of Designs from An Automatic Logic Generator", by T. D. Friedman et al., IEEE 7th DA Conference, 1970, pp. 71-89. --.

"Trevillyan-Trickey, H., Flamel: *A High Level Hardward Compiler*, IEEE Transactions On Computer Aided Design, Mar. 1987, pp. 259-269." should be -- Trevillyan-Trickey, H., Flamel: *A High Level Hardware Compiler*, IEEE Transactions On Computer Aided Design, Mar. 1987, pp. 259-269. --.

In the abstract:

Every occurrence of "functional architecture independent" should be -- architecture independent functional --.

Column 1, line 19, "a" should be -- an --.

UNITED STATES-PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 2, line 10, "functional architecture independent" should be -- architecture independent functional --.

Column 2, line 21, "functional architecture independent" should be -- architecture independent functional --.

Column 2, lines 29-30, "functional architecture independent" should be -- architecture independent functional --.

Column 2, line 31, "structural" should be after "specific".

Column 3, lines 51-52, "representation" should be after "architecture independent".

Column 3, lines 61-62, "integrated" should be after "specific".

Column 6, line 62, after "22" insert -- . --.

Column 7, line 43 (in Table 1), "C = A B" should be -- C = A^B --.

Column 8, line 9 should end with the word "flowchart" and "history" should begin on the next line.

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432 Page 3 of 4

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 10, line 23, "data paths" should be -- datapaths --.

Column 10, line 68, delete "The rule format to be used is as follows:".

Column 12, line 54, "Enqineering" should be -- Engineering --.

Column 13, line 55, "block" should be -- blocks --.

In the Claims:

Column 14, line 68, before "means" (first occurrence) insert -- specification --; after "means" (second occurrence) delete "specification".

Column 15, line 9, before "means" (first occurrence) insert -- specification --; after "means" (second occurrence) delete "specification".

Column 15, line 35, after "circuit" insert -- , --.

Column 15, line 36, "marco" should be -- macro --.

Column 15, line 49, "form" should be -- from --.

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 4,922,432 Page 4 of 4

DATED : May 1, 1990

INVENTOR(S) : Hideaki Kobayashi, et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 16, line 14, "condition" should be -- conditions --.

Column 17, line 19, after "base" insert -- a --.

Signed and Sealed this

Fourteenth Day of January, 1992

Attest:

Attesting Officer

HARRY F. MANBECK, JR.

Commissioner of Patents and Trademarks

PTO UTILITY GRANT
Paper Number 12

The United States of America

The Commissioner of Patents and Trademarks

Has received an application for a patent for a new and useful invention. The title and description of the invention are enclosed. The requirements of law have been complied with, and it has been determined that a patent on the invention shall be granted under the law.

Therefore, this

United States Patent

Grants to the person or persons having title to this patent the right to exclude others from making, using or selling the invention throughout the United States of America for the term of seventeen years from the date of this patent, subject to the payment of maintenance fees as provided by law.

Acting Commissioner of Patents and Trademarks

Attest

RCL000265

EXHIBIT 6

United States Patent [19]

Darringer et al.

[11] **Patent Number: 4,703,435**

[45] **Date of Patent: Oct. 27, 1987**

[54] **LOGIC SYNTHESIZER**

[75] Inventors: **John A. Darringer**, Mahopac; **William H. Joyner, Jr.**, Katonah, both of N.Y.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **631,364**

[22] Filed: **Jul. 16, 1984**

[51] **Int. Cl.**[4] **G06F 7/00**; G06F 15/60

[52] **U.S. Cl.** **364/489**; 364/300; 364/488

[58] **Field of Search** 364/488, 489, 490, 491, 364/300; 307/303

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,377,849	3/1983	Finger et al.	364/491
4,580,228	4/1986	Noto	364/300 X
4,591,993	5/1986	Griffin et al.	364/491
4,612,618	9/1986	Pryor et al.	364/488 X

FOREIGN PATENT DOCUMENTS

0168650 1/1986 European Pat. Off. .

OTHER PUBLICATIONS

Friedman et al., "Methods used in an Automatic Logic Design Generator (Alert)", *IEEE Trans. Comp. C*-18, pp. 593–614, 1969.

Mitchell et al., "The Use of High Speed Proms to Generate Boolean Functions,", *Wescon Technical Papers*, Sep. 1978, pp. 1–7.

Mano, "Digital Logic and Computer Design", Ch. 3, pp. 72–103, 1979.

Introduction to the Automated Synthesis of Computer; Herbert Schorr; Department of Electrical Engineering Digital Systems.

Laboratory Technical Report No. 16 (Mar. 1962). Ph.D. Thesis, Princeton University.

Minimization of Boolean Functions; F. J. Hill et al.; "Introduction to Switch Theory and Logical Design", 1973.

The Description, Simulation, and Automatic Implementation of Digital Computer Processors; John A. Darringer.

The Experimental Compiling System; F. E. Allen; IBM J. Res. Development; vol. 24, No. 6, Nov. 1980.

Logic Synthesis Through Local Transformation; John A. Darringer; IBM Journal of Research and Development; vol. 25, No. 4, Jul. 1981.

Programming Language; Yaohan Chu; vol. 8, No. 10, 10/65.

Development and Application of a Designer Oriented Cyclic Simulator; G. J. Parasch; 13th DA Conference 1976.

Logic Synthesis; Melvin A. Breuer; M. Breuer "Design Automation of Digital Systems", Prentice-Hall 1972.

Synthesis of Combinational Logic Networks; D. L. Dietmeyer "Logic Design of Digital Systems"; Allyn & Bacon, Boston 1978.

Quality of Designs from an Automatic Logic Generator (Alert)*; Theodore D. Friedman and Sih-Chin Yang; 7th DA Conference 1970.

On Logic Comparison; Leonard Berman; 18th DA Conference 1981.

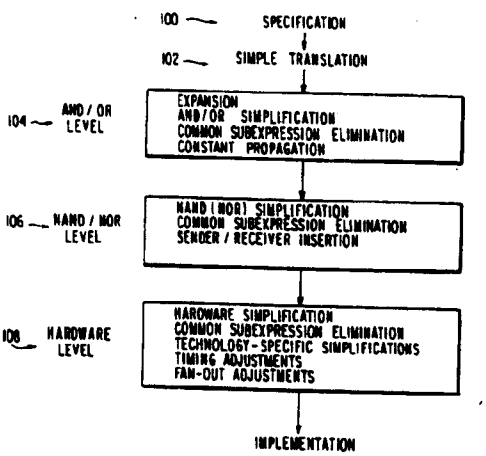
(List continued on next page.)

Primary Examiner—Errol A. Krass
Assistant Examiner—Joseph L. Dixon
Attorney, Agent, or Firm—Sughrue, Mion, Zinn, Macpeak, and Seas

[57] **ABSTRACT**

Logic is synthesized from a flowchart-level description by first generating an AND/OR logic design, simplifying the AND/OR logic, converting the AND/OR logic to NAND or NOR logic, applying particular sequences of simplifying transformations to the NAND or NOR logic, converting the simplified NAND or NOR logic to a target technology, and simplifying the target technology where possible. The end result is an interconnection of primitives of the target technology in a language from which automated logic diagrams can be produced.

24 Claims, 33 Drawing Figures



OTHER PUBLICATIONS

Automated Exploration of the Design Space for Register-Transfer (RT) Systems: Barbacci Mario Roberto;

The Design and Analysis of an Automated Design Style Selector: Thomas Donald Earl, Jr.

Automation of Module Set Independent Register--Transfer Level Design: Edward Alfred Snow, III.

A new Look at Logic Synthesis; John A. Darringer; 17th DA Conference 1980.

Experiments in Logic Synthesis; John A. Darringer; IEEE ICCC 1980.

Methods used in an Automatic Logic Design Generator (Alert); Theodore D. Friedman; IEEE Transactions on Computers; vol. C-18, No. 7, Jul. 1969.

Register-Transfer Level Digital Design Automation: The Allocation Process; Louis Hafer; 15 DA Conference 1978.

The CMU Design Automation System; An Example of Automated Data Path Design; A. Parker et al.; 16 DA Conference 1979.

Lores-Logic Reorganization System; Shunichiro Nakamura et al.; 15 DA Conference 1978.

Translation of a DDL Digital System Specification to Boolean Equations; James R. Duley and Donald L. Dietmeyer, IEEE Trans. on Comp. vol. C-18. No. 14, '69.

DDL-A Digital System Design Language; James Robert Duley; Ph.D. 1967.

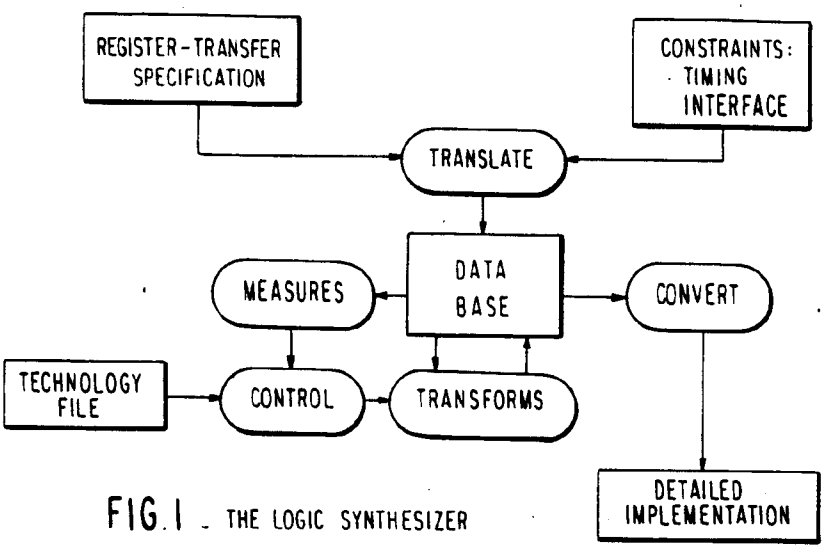


FIG. 1 - THE LOGIC SYNTHESIZER

FIG. 2

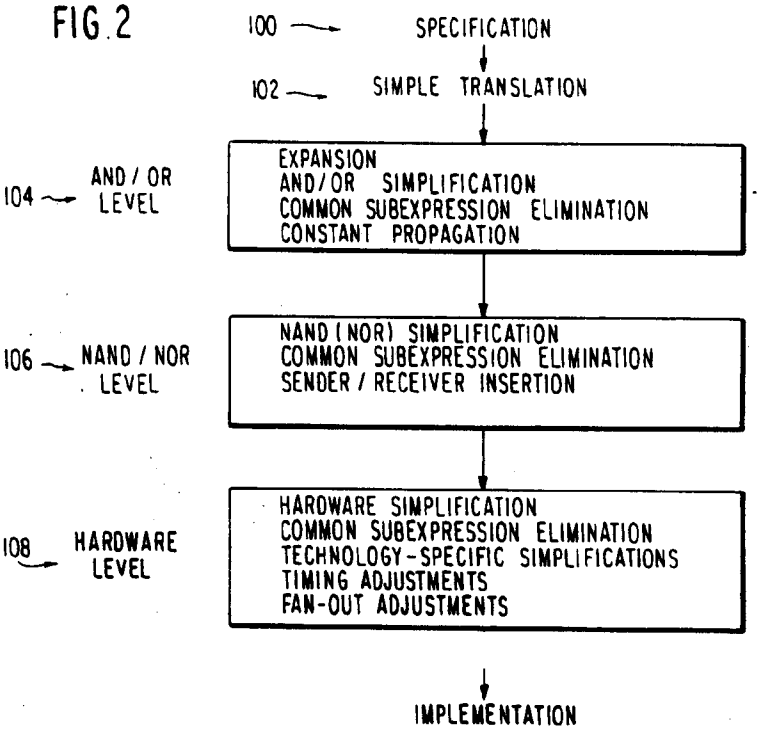


FIG. 3a

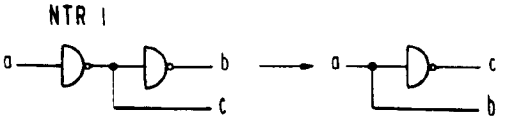


FIG. 3b

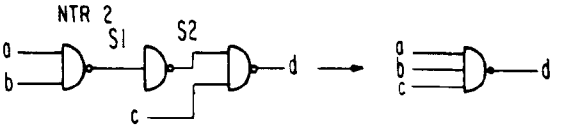


FIG. 3c

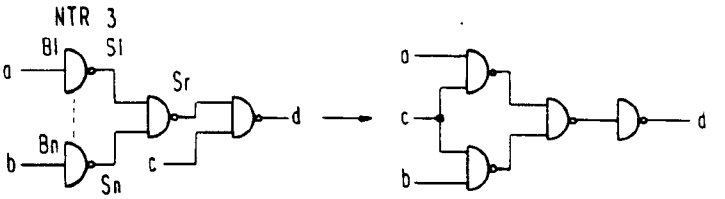


FIG. 3d

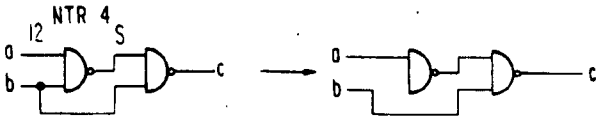


FIG. 3e

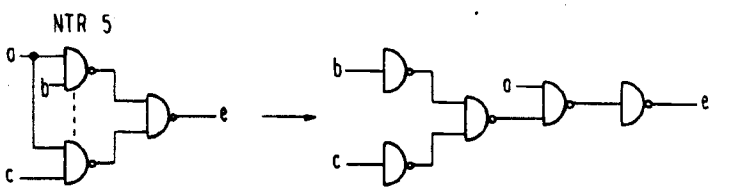


FIG. 3f

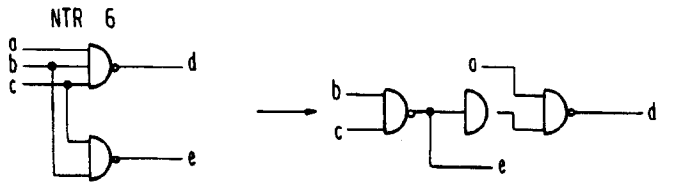


FIG. 3g

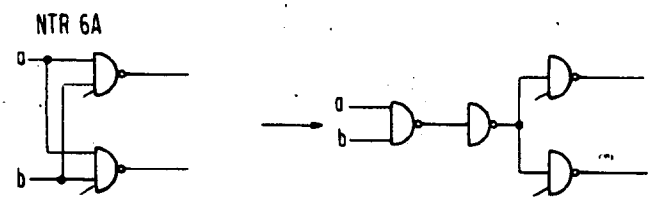


FIG. 3h

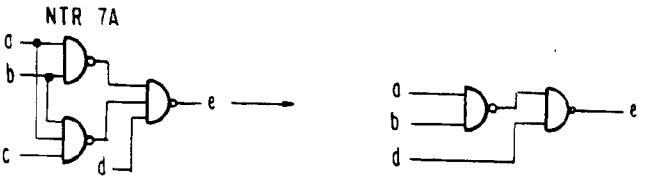


FIG. 3i

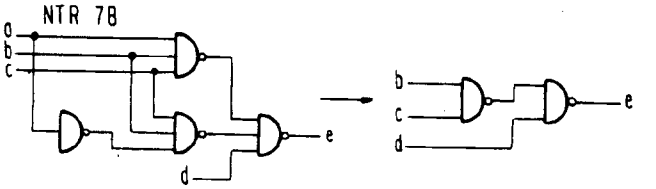


FIG. 3j

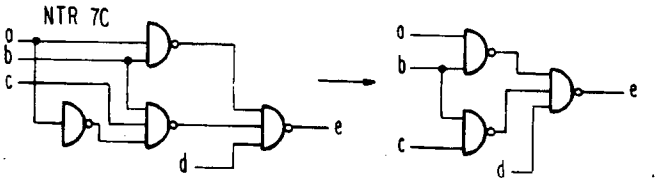


FIG. 3k

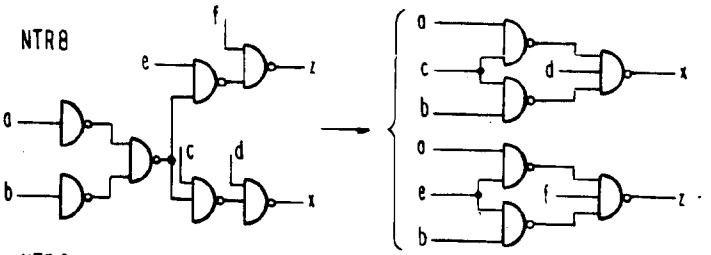


FIG. 3l

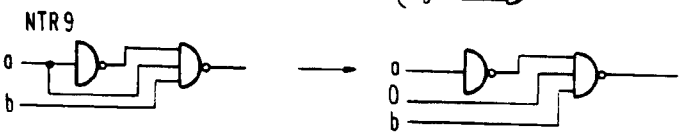


FIG. 3m

FIG. 3n

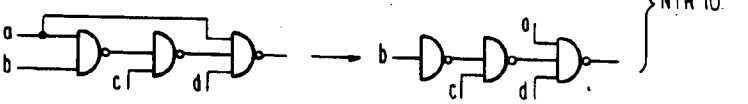


FIG. 3o

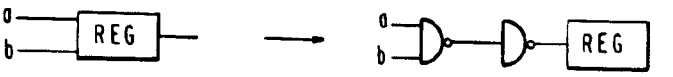


FIG. 3p

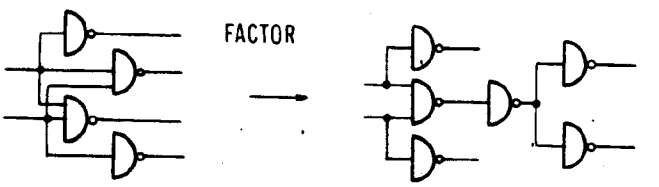


FIG. 4

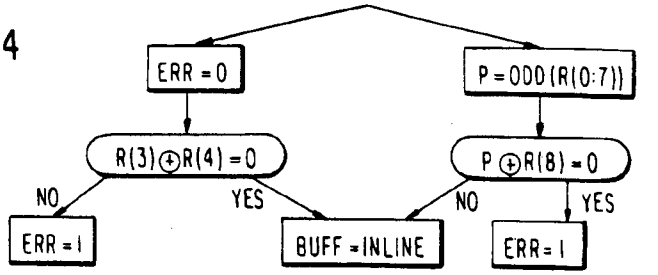


FIG. 5a
ELIMINATE COMMON TERMS

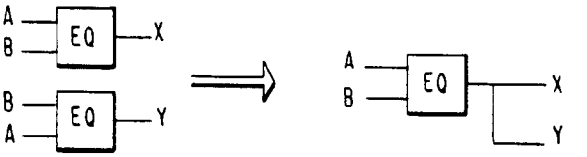


FIG. 5b
SIMPLIFY PARITY OPERATORS

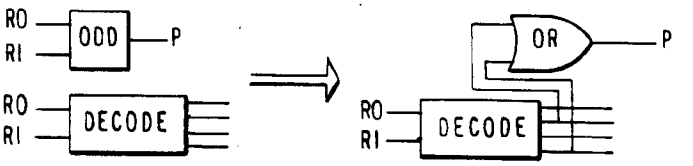


FIG. 6

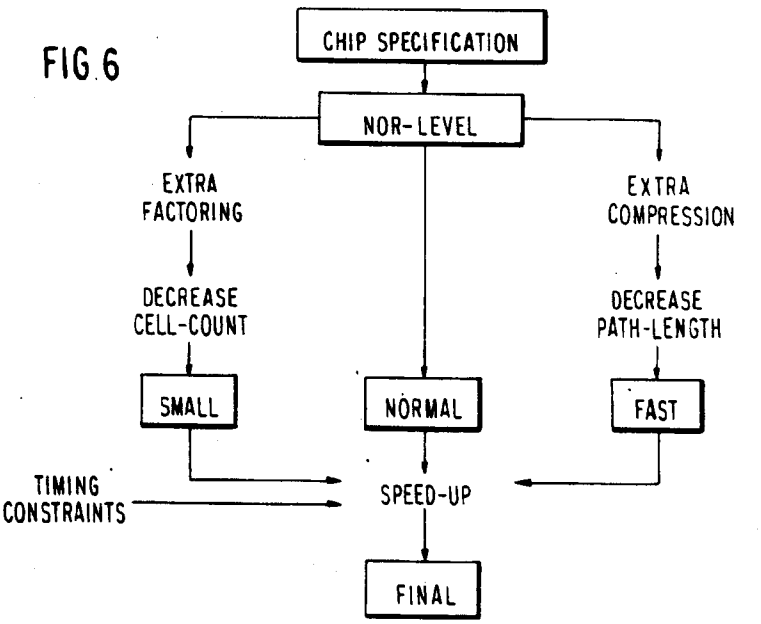


FIG.7a
MERGE RECEIVERS

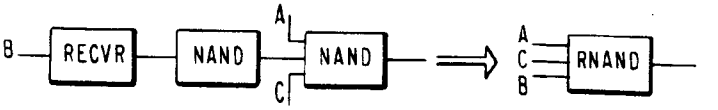


FIG.7b
INSERT DOTS

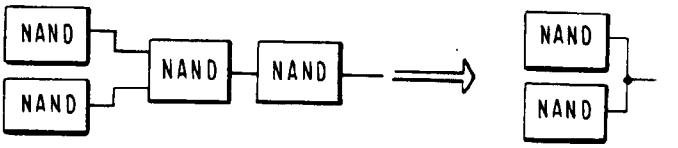


FIG.7c

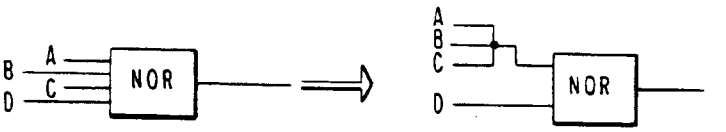


FIG.8a

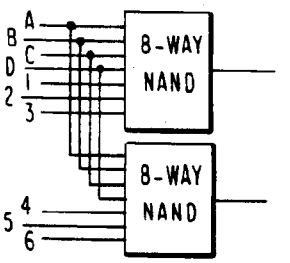


FIG.8b

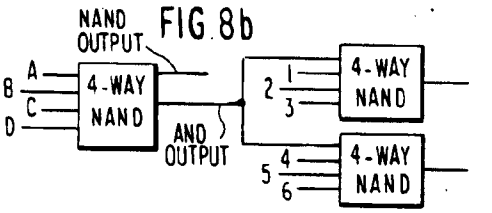


FIG.9a

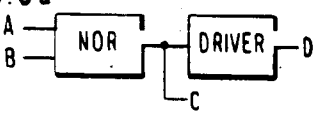
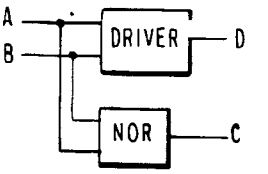


FIG.9b

FIG.9c



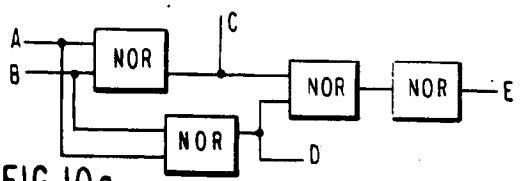


FIG.10a

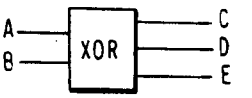


FIG.10b

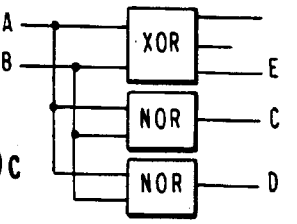


FIG.10c

LOGIC SYNTHESIZER

BACKGROUND OF THE INVENTION

This invention is directed to logic design, and more particularly to a method of automated logic design.

As the complexity of processors has increased, the task of processor logic design has become more difficult. The designer may begin by designing a flow chart or other register-transfer level description to describe the intended operation of the processor, and the processor operation is then simulated from this description in order to ensure that a processor operating in accordance with the flow chart will provide the desired results. A logic implementation is then designed to achieve the operation described in the flow chart, and the resulting logic diagram and original flow chart specification are compared to ensure consistency. Finally, a physical layout is designed in accordance with the logic diagram implementation.

The above process has become significantly more difficult and extraordinarily time consuming with the increasing complexity of the processors being designed. For example, each chip in the 3081 processor available from International Business Machines Corporation includes over 700 circuits capable of performing extremely complex functions. The flow chart specification of such a processor will be quite complex, and even a first attempt at a logic diagram implementation will require a substantial amount of time. Further, with increasing processor complexity, the competing interests of gate count and timing constraints become increasingly difficult to satisfy. More particularly, a typical timing constraint may be that a signal must be provided from the output of register A to the input of register B within some predetermined period of time, and the designer may first propose a logic arrangement intended to satisfy this timing constraint while using a minimal number of gates in the circuit path between registers A and B. After timing analysis, however, it may be discovered that the timing constraint has not been satisfied, and the designer must then revise the arrangement of logic between the registers A and B, e.g., by using a larger number of gates to improve the processing speed in that area. Several iterations of design may be required before a logic design is obtained which indeed satisfies all timing constraints with the minimum gate count, and it is therefore not uncommon for the logic design to be quite costly in terms of engineering time.

In view of the above, there has been significant recent activity in the field of automatic logic synthesis. Early work centered on developing algorithms for translating a boolean function into a minimum 2-level network of boolean primitives, and extensions were developed for handling limited circuit fan-in and alternative cost functions. However, because these algorithms employ 2-level minimization, the time required to implement these algorithms increases exponentially with the number of circuits. The use of such algorithms therefore becomes impractical in designing large processors.

Other efforts have attempted to raise the level of specification, e.g., by beginning with behavioral specifications and producing technology-independent implementations at the level of boolean equations. However, the results of such techniques were usually more expensive than manual implementations and did not take advantage of the target technology. For example, the system described by T.D. Friedman et al, in "METHODS USED IN AN AUTOMATIC LOGIC DESIGN GENERATOR (ALERT)," IEEE Trans. on Computers, Vol. C-18, No. 7 pp. 593–614 (1969), produced an implementation for an IBM 1800 processor which required 160% more gates than the manual design for that processor. Several attempts have been made to produce more efficient logic and to give the designer more control over the implementation, e.g., as described by: H. Schorr, "Toward the Automatic Analysis and Synthesis of Digital Systems," Ph.D. Thesis, Princeton University, Princeton, NJ, 1962; C.K. Mestenyi, "Computer Design Language Simulation and Boolean Translation," *Technical Report* 68–72, Computer Science Department, University of Maryland, College Park, MD, 1968; F.J. Hill and G. R. Peterson, *Digital Systems: Hardware Organization and Control*, John Wiley & Sons, Inc., New York, 1973. However, this control has resulted in specification language constraints, so that the specification is at a fairly low level and in closer correspondence with the implementation. This necessarily decreases the advantage of an automated approach, bringing it closer to a system for logic entry rather than logic synthesis.

Several tools have been developed to support the early part of the design cycle, e.g., as described in: M. Barbacci, "Automated Exploration of the Design Space for Register Transfer Systems," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1973; D. E. Thomas, "The Design and Analysis of an Automated Design Style Selector," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1977; E. A. Snow, "Automation of Module Set Independent Register-Transfer Level Design," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1978; L. J. Hafer and A. C. Parker, "Register-Transfer Level Digital Design Automation: The Allocation Process," *Proceedings of the Fifteenth Design Automation Conference*, Las Vegas, NV, 1978, pp. 213–219; A. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive, and J. Kim, "The CMU Design Automation System - An Example of Automated Data Path Design," *Proceedings of the Sixteenth Design Automation Conference*, Las Vegas, NV. 1978, pp. 73–80. The technique described in the last-cited publication began with a functional description of a machine and produced an implementation in two technologies of the registers, register operators and their interconnections, but not the control logic to sequence the register transfers. For both TTL and CMOS implementations, however, the automated implementation required substantially more chip area than existing manual designs.

There has also been recent work in logic remapping, i.e., transforming existing implementations from one technology to another. S. Nakamura et al S. Nakamura, S. Murai, C. Tanaka, M. Terai, H. Fujiwara, and K. Kinoshita, "LORES-Logic Reorganization System," *Proceedings of the Fifteenth Design Automation Conference*, Las Vegas, NV, 1978, pp. 250–260; describe a system which will help a dealgner translate an existing small-or medium-scale integration implementation into large-scale integration. However, remapping usually involves one-to-one substitution of new technology primitives for old technology primitives, and this often fails to take advantage of simplification which may be available at a higher technology-independent level.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide an automated logic synthesis technique which overcomes the above-described drawbacks. It is a more particular object of the present invention to provide such an automated logic synthesis technique which is capable of operating at a relatively high speed while achieving end results comparable to those obtained by manual design. It is a still further object of this invention to provide such an automatic logic synthesis technique capable of achieving satisfactory results in a number of different technologies.

Briefly, these and other objects of the invention are achieved by a logic synthesis method in which a register-transfer level flowchart specification is translated in a straightforward manner into a simple AND/OR logic implementation. After expanding the logic implementation to elementary representation and then applying textbook simplifications, the simplified AND/OR implementation is translated to a NAND or NOR implementation, depending on the target technology. The NAND or NOR implementation is then simplified by applying a sequence of simplification transformations which have been found by the present inventors to achieve satisfactory results, with the transformation sequence being modified to achieve "normal," "fast" or "small" logic designs. After simplification at the NAND/NOR level, the logic implementation is then translated to the target technology and further simplified. The result is an interconnection of the primitives of the target technology in a language from which automated logic diagrams can be produced in a known manner, and which can be submitted to existing programs for automated placement and wiring and chip fabrication.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be more clearly understood from the following description in conjunction with the accompanying drawings, wherein:

FIG. 1 is a conceptual diagram of the logic synthesis technique according to the present invention;

FIG. 2 is a chart illustrating the multiple levels of simplification in the logic synthesis technique according to the present invention;

FIGS. 3(*a*)–3(*p*) illustrate simplifying transformations at the NAND/NOR level;

FIG. 4 is a simple illustration of a portion of a flowchart specification from which the present invention begins;

FIGS. 5(*a*)–(*b*) illustrate simplifications which may be performed at the AND/OR level;

FIG. 6 is a diagram illustrating the different scenarios of simplification at the NAND/NOR level;

FIG. 7(*a*)–7(*c*) illustrate examples of simplification at the hardware level; and

FIGS. 8(*a*)–8(*b*), 9(*a*)–(*c*) and 10(*a*)–10(*c*) illustrate further examples of technology-specific hardware simplifications.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The logic synthesis method according to the present invention is generally illustrated in FIG. 1. Previous publications describing some aspects of the system according to this invention, all of which are incorporated herein by reference, are: J. A Darringer and W. H. Joyner, "A New Look at Logic Synthesis," *Proceedings of the Seventeeth Design Automation Conference* Minneapolis, MN, 1980, pp. 543–549; J.A. Darringer, W. H. Joyner, L. Berman, and L. Trevillyan, "Experiments in Logic Synthesis," *Proceedings of the IEEE International Conference on Circuits and Computers ICCC*80, Port Chester, NY, 1980, pp. 234–237A; J. A. Darringer, W. H. Joyner, C. L. Berman and L. Trevillyan, "Logic Synthesis Through Local Transformations," *IBM Journal of Research and Development, Vol.* 25 No. 4, July 1981.

The present invention is an automatic replacement for part of the manual design process. It operates on a logic design at three levels of abstraction. It begins with an initial implementation generated in a straightforward manner from the specification. The implementation can be simplified at this level, and then moved to the next level. This simplification is accomplished by transformations, either locally or globally, to achieve the simplification or refinement. By being able to operate on the implementation at several levels, the system can often make a small change at one level that will cause a larger simplification at a lower level. By using function-preserving transformations, it is ensured that in all cases the implementation produced will be functionally equivalent to the specified behavior. The inputs to the system illustrated in FIG. 1 are a description, in a register-transfer level, flow chart-control language, of logic functions to be implemented on a chip in a specified master slice technology, together with the interface constraints and a technology file which characterizes the target technology. The output of the system is a detailed interconnection of the primitives of the target technology in a language from which automated logic diagrams (ALD's) may be produced and which can be submitted to existing programs for automated placement and wiring and chip fabrication. The output implementation is in terms of the target technology and satisfies technology-specific constraints. Some timing or other physical problems may not be detectable before placement and wiring, and in such cases the synthesis process is repeated with a revised specification or modified constraints until an acceptable implementation is achieved.

The method according to this invention comprises PL/I programs operating on a representation of the logic in a data management system. The data management system is preferably that described by F. E. Allen et al, "THE EXPERIMENTAL COMPILING SYSTEM," IBM Journal of Research and Development, Volume 24 (1980), pages 695–715. The logic synthesis data base uses a single organization component referred to as a "box," with each box having input and output terminals which are connected by wires to other boxes. Each box also is designated by a type, which may be a primitive or may reference a definition in terms of other boxes. Thus, a hierarchy of boxes can be used, and an instance of a high-level box such as a parity box can be treated as a single box or expanded into its next-level implementation when that is desirable.

The logic synthesis data base is made of two groups of tables. The first group describes the technology being used, and is created from a technology file containing, for each box type, information such as name, function and number and names of input and output pins. These data are created in batch mode and read during initialization of the interactive system.

The second group of tables contains the representation of the logic created by the system. This group consists of a box table, a signal table and a set of auxiliary tables which describe the relationship between the boxes and the signals. There is some intentional redundancy in the data, i.e., each box has a complete list of input and output signals and each signal has a source and a list of sinks. Every box table entry contains type information which provides a link to the technology group, thus allowing programs to obtain technology information about a specific box.

Using the system generally illustrated in FIG. 1, a synthesis process according to the present invention may follow the sequence of steps shown in FIG. 2. FIG. 2 illustrates the three essential levels of description used in the method of the present invention: the initial AND-/OR level 104, a NAND or NOR level 106 (depending on the target technology), and a hardware level 108 in which the types of the boxes are books or primitives of the target technology. At every level, the implementation is a network of boxes connected by signals. The purpose of this type of implementation is to find a set of transformations and a sequence of applying these transformations such that the original functional specification could be transformed by a sequence of small steps into an acceptable implementation.

As pointed out above, the process of this invention begins at step 100 with a register-transfer level description e.g. of the type shown in FIG. 4. The description consists of two parts: a specification of the inputs, outputs and latches of the chip to be synthesized; and a flowchart-like specification of control, describing for a single clock cycle of the machine how the chip outputs and latches are set according to the values of the chip inputs and previous values of the latches. At step 102 in FIG. 2, the register-transfer level description undergoes a simple translation to an initial implementation of AND/OR logic. This AND/OR level is produced by merely replacing specification language constructs with their equivalent AND/OR implementations in a well known manner, e.g., as described in J. R. Duley, "DDL - A Digital Design Language," Ph.D. Thesis, University of Wisconsin, Madison, WI, 1968; or J. A. Darringer, "The Description, Simulations and Automatic Implementation of Digital Computer Processors," Ph. D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1969. At this first level 104 in FIG. 2, the logic begins in the form of an interconnection of boxes designated by types representing the operations which the perform, e.g., AND, OR, NOT, PARITY, EQ, XOR, DECODE, REGISTER (generic latch), SENDER, RCVR. At step 104 in FIG. 2, the initial AND/OR implementation is first expanded by taking all operators more complex than AND, OR or NOT and replacing these more complex operators with combinations of AND, OR, and NOT. Beginning with this expanded AND/OR logic, simplification is achieved by invoking PL/I program transformations which search for patterns of interconnected primitives and replace them by functionally equivalent patterns which are simpler in that they use fewer instances of operators, fewer connections, etc. The transformations at the AND/OR level 104 are local, textbook simplifications of boolean expressions, most of these simplifications reducing the number of boxes but not producing a normal form. Examples of simplifications are shown in FIGS. 5(*a*) and 5(*b*). Some of these transformations are similar to optimizing compiler techniques, e.g., constant propagation (moving "0" or "1" signals through logic blocks), common term elimination (combining blocks which compute the same function), combining nested associativecommutative operators, eliminating single input AND's and OR's, etc. Further examples of transformations used are as follows:

NOT(NOT(a))→a
AND(a, NOT(a))→0
OR(a, NOT(a))→1
OR(a, AND(NOT(a),b)→OR(a,b)
XOR(PARITY(a_1, . . ., a_n), b)→
PARITY(a_1, . . ., a_n, b)
AND (a, 1)→a
OR(a, 1)→1

These transformations may leave fragments of logic disconnected, and this can be cleaned up in a manner similar to the way in which compilers perform dead-code elimination.

After simplification at the AND/OR level 104, the simplified AND/OR implementation is transformed into a NAND or NOR implementation. Whereas AND-/OR logic requires the use of multiple different operators in a logic design, NAND or NOR logic requires fewer operators, i.e., in a NAND logic design all logical functions can be implemented using some combination of only NAND gates. Whether a NAND or NOR implementation is produced is dependent upon the primitives available in the target technology. However, the NAND or NOR description is not technology-specific, in that there are no fan-in or fan-out restrictions. (Fan-in refers to the number of signals coming into a box, and fan-out refers to the number of sinks or destinations of a signal.) The transition to these primitives is accomplished naively by local transformations and may introduce unnecessary double NANDs or NORs which will be eliminated later. Also at this point, the chip interface information is used to place generic, i.e., not technology-specific, senders and receivers on the chip inputs and primary outputs, and to insert inverters where necessary to ensure the correct signal polarities. Techniques for accomplishing this transformation are well-known and need not be described here in detail.

At step 106 in FIG. 2, simplifying transformations are applied to each signal in the network. The NAND and NOR transforms are more difficult, and extensive experiments by the present inventors at the NAND/NOR level have resulted in a sequence or "scenario" of transformations which will produce acceptable results. The transformations are local in that they replace a small subgraph of the network (usually five or fewer boxes) with another subgraph which is functionally equivalent but simpler according to some measure. These transformations attempt to reduce the number of boxes of the implementation without increasing the number of connections. To accomplish this, the transformations must check the fan-out of the various signals involved, since this will affect the number of boxes and signals actually removed. Some of the transformations attempt to remove reconvergent fan-out which contributes to untestable stuck faults.

Some of the transformations are applied throughout the network in a number of iterations, preferably until no more transformations apply. FIGS. 3(*a*)-3(*n*) illustrate the NAND transformations NTR1 thru NTR10 used in one embodiment of this invention, and the NOR transformations would be identical except for the operator. Each transformation has an associated condition that determines if the replacement will simplify the

implementation by reducing boxes or connections. These conditions depend on the fan-out of the intermediate signals and on whether the target technology is assumed to have dual-rail output.

Experiments with the NAND/NOR level transformations have resulted in a normal sequence or "scenario," of transformations which have produced acceptable results. A "fast" scenario was developed which resulted in shorter path lengths, and a "small" scenario was also developed to obtain smaller designs. These are generally indicated in FIG. 6. In the preferred embodiment of this invention, the sequence of steps in the normal NAND/NOR scenario would be as follows:

APPLY GENNOR: (or APPLY GENNAND);
UNTIL NOCHANGE APPLY NTR1, NTR2, CLEANUP, NTR3, NTR4, NTR10, CLEANUP, NTR7, NTR9, PROPCON, CLEANUP, CTE, CLEANUP;
FANIN 4;
APPLY NTR6A, FACTORN, NTR6A, CLEANUP;
APPLY NTR10, CLEANUP, NTR7 (NOINCREASE), NTR9, PROPCON, CLEANUP:
APPLY CTE, CLEANUP; FANIN 8:
APPLY NFANIN, NTR11, CLEANUP;

The GENNOR or GENNAND transformations merely transform the AND/OR implementation into either NAND or NOR logic in accordance with the target technology. This type of transformation is well understood in the art and need not be described in detail here.

NTR1 in FIG. 3(*a*) removes double inverters and always applies, since it is always considered desirable to reduce the number of cells, and because this transformation does not increase connects or path lengths. This transform, and others, may in some instances increase fan out, but the fan out can be reduced, if necessary, at a later point.

NTR2 in FIG. 3(*b*) applies only if s_1 has no fan out and s_2 fans out only to primitives, i.e., either NANDs or NORs. This transform will not apply if it will result in an increase in the number of connects. For example, in the transformation illustrated in FIG. 3(*b*), gates 10 and 12 are eliminated and their corresponding input and output connections are also eliminated. However, if s_2 fans out to four NANDs, it would be necessary to apply the NTR2 transformation to each one, resulting in an increase in the number of connects.

NTR3 in FIG. 3(*c*) applies only if none of the gate outputs s_i fans out, s_r does not fan out, and no gate B_i exceeds the fan-in threshold for a single-cell book. This helps set up later dotting.

NTR4 in FIG. 3(*d*) removes redundancy locally. Redundancy is a property of a combinational logic circuit, and is present when the network contains a signal that can be set to a constant value without changing the function of that network. NTR4 also replicates logic if the output s of gate 12 fans out.

NTR6A in FIG. 3(*g*) sets up dotting and is only run if dotting is allowed in the target technology.

NTR7 eliminates some forms of redundant connections. This transform will replicate boxes, if necessary, unless the parameter NOINCREASE is specified. NTR7 actually comprises three transforms illustrated in FIGS. 3(*h*)–3(*f*), all of which are run each time NTR7 is called for in the above program.

NTR9 in FIG. 3(*i*) handles cases where a signal and its negation both go to a NOR or NAND gate. The "0" input to gate 14 will be a "1" for the equivalent NOR transformation. This transform should be followed by PROPCON, described below.

NTR10 includes two different transforms illustrated in FIGS. 3(*m*) and 3(*n*), both of which are run each time NTR10 is called for. The NTR1O transform is run only if the outputs of gates 18 and 20 and FIG. 3(*n*) do not fan out.

NTRll in FIG. 3(*o*) makes all generic registers (considered to have the OR function) have a fan-in of 1 by preceding each register with an appropriate number of primitives.

PROPCON, CLEANUP and CTE are analogous to the compiler operations of constant propagation elimination, dead-code elimination and common sub-expression elimination, respectively. Common sub-expression elimination, or common term elimination, refers to locating boxes which produce the same logic value, eliminating one box, and sharing the output of the other box.

FANIN 4 does not in itself perform any transformation but instead sets a variable known as "FANIN" to a value of 4.

FACTORN examines only boxes exceeding the FANIN limitations specified by the variable FANIN. It then applies the transformation of FIG. 3(*p*). This transformation will not reduce all boxes to below the specified FANIN limit, but only those boxes to which it applies by finding common sinks.

NFANIN corrects the fanin to the specified limit by building fanin trees which it constructs to have the fewest boxes and then to lengthen as few paths as possible.

In a NOCHANGE loop, the transformations are repeatedly run in their specified order until no further change in the logic occurs. In general, the order of the transformations and their inclusion in the NOCHANGE loop is such that suceeding transformations are invoked when preceding transformations can cause them to apply. For example, in the first loop, the sequence beginning with NTR9 is used to remove gates having complementary inputs. Since this can produce constant zeros or ones, constant propagation (PROPCON), removal of unconnected boxes (CLEANUP), common term elimination (CTE), and then more CLEANUP (to deal with now-unconnected common terms) must be run. On the other hand, after fan-in correction by factoring and NFANIN, some transformations should not be run, because they may destroy the fan-in limits already enforced.

In looking again at the program above, it can be seen that certain sequences of functions are performed, with some functions comprising a plurality of transformations. More particularly, with regard to the first NOCHANGE loop, transformations NTR1, NTR2, CLEANUP, NTR3 operate to reduce logic depth, i.e., number of levels of logic from input to output, with NTR1 reducing logic depth from two levels to one and NTR2 reducing logic depth from three levels to one. NTR3 at first glance appears to provide no depth reduction, since it transforms three levels of logic to three levels of logic. However, in some instances the last level, gate 11, can be subsequently eliminated, so that NTR3 is often useful in reducing logic depth.

Reducing logic depth, i.e., .compressing the logic into fewer levels, will increase the chance of detecting redundancy. Thus, NTR4, NTR10, CLEANUP, NTR7, NTR9, PROPCON, CLEANUP applied to remove redundancy.

After removing redundancy, a common terms elimination sequence CTE, CLEANUP is run.

After the NOCHANGE loop has finished running, transformations are applied to introduce dot patterns and to reduce fan-in to a specific level. This is accomplished by the step FANIN 4 which sets the fan-in limit to a value of 4, followed by the sequence NTR6A, FACTORN, NTR6A, CLEANUP, which serves to reduce fan-in at the expense of logic depth.

Once again, the introduction of dot patterns and the factoring to reduce fan-in may result in redundancy, so that the redundancy removal sequence NTR10, CLEANUP, NTR7, NTR9, PROPCON, CLEANUP is applied.

Common terms are then eliminated by running CTE, CLEANUP.

Finally, the logic must be adjusted to the maximum fan-in value permitted by the target technology, e.g., a fan-in value of 8. This is achieved by applying FANIN 8 to set the fanin value at 8 followed by NFANIN, CLEANUP.

As should now be appreciated, the above program can be functionally represented as follows:

A. LOGIC DEPTH REDUCTION LOOP
  A1. REDUCE LOGIC DEPTH
  A2. REMOVE REDUNDANCY
  A3. ELIMINATE COMMON TERMS
B. INTRODUCE DOT PATTERNS AND FACTOR TO REDUCE FANIN TO SPECIFIC LEVEL
C. REMOVE REDUNDANCY
D. ELIMINATE COMMON TERMS
E. ADJUST LOGIC TO MAXIMUM PERMITTED FANIN

The operations subsequent to the logic depth reduction loop may tend to expand the logic depth, so that the above process can generally be seen as a compression stage followed by an expansion stage. While it may be theoretically possible to obtain maximum logic depth reduction through two-level boolean minimization, this would compress the logic so far that re-expansion to take advantage of other simplifying transformations, e.g., at the subsequent hardware simplification, would be much more difficult. Thus, the logic compression transforms have been found particularly suitable.

The program set forth above concerns a normal scenario, and the "fast" and "small" scenarios can be obtained by modifying the above program as follows: for the small scenario, the following additional NOCHANGE loop is inserted after the NOCHANGE loop in the normal scenario:

UNTIL NOCHANGE APPLY NTR6, NTR5, NTR1, NTR2, CLEANUP, NTR3, NTR4, NTR10, CLEANUP, NTR7, NTR9, PROPCON CLEANUP, CTE, CLEANUP;

NTR5 in FIG. 3(*e*) applies only if the number of cells does not increase, and NTR6 in FIG. 3(*f*) applies only if the number of cells is decreased. Inspection of NTR5 and NTR6 shows that they can increase path length, and they are consequently only used in the small scenario. The other transformations in the added loop are provided to act on any changes which may result from NTR5 and NTR6. For example, NTR5 and NTR6 can produce double inverters, so the sequence beginning with NTR1 is run. NTR1 eliminates double inverters, and can introduce situations where other transforms apply.

Examination of the second NOCHANGE loop set forth above will reveal that the loop includes a first sequence NTR6, NTR5 for reducing the cell count by increasing the logic depth. The sequence NTR1, NTR2, CLEANUP, NTR3 is then applied to mitigate the logic depth reduction by taking advantage of transforms made available by NTR6, NTR5. After this logic depth reduction sequence, the redundancy removal and common term elimination sequence are applied in the first NOCHANGE loop.

Thus, the program for the "small" scenario can be written:

A. LOGIC DEPTH REDUCTION LOOP
  A1. REDUCE LOGIC DEPTH
  A2. REMOVE REDUNDANCY
  A3. ELIMINATE COMMON TERMS
  A'. CELL COUNT REDUCTION LOOP
  A1'. REDUCE CELL COUNT
  A2'. REDUCE LOGIC DEPTH
  A3'. REMOVE REDUNDANCY
  A4'. ELIMINATE COMMON TERMS
B. INTRODUCE DOT PATTERNS AND FACTOR TO REDUCE FANIN TO SPECIFIC LEVEL
C. REMOVE REDUNDANCY
D. ELIMINATE COMMON TERMS
E. ADJUST LOGIC TO MAXIMUM PERMITTED FANIN

While the "small" scenario is designed to emphasize minimization of gate count, the "fast" scenario is designed to emphasize shorter path lengths, sometimes at the expense of gate count. Path length refers to the delay along a path from a signal's source to one of its destinations. Usually, path lengths are measured from registers or primary chip inputs to registers or primary chip outputs. The result can be the number of boxes in the path or the estimated delay of that path in nanoseconds.

The fast scenario inserts a call to NTR8 as the last step run in the first NOCHANGE loop. Immediately thereafter, FANIN is set to a value of 8 rather than 4, and NTR11 is omitted from the last line of the program. The significance of these changes to the program is as follows:

NTR8 in FIG. 3(*k*) is used in the fast scenario because it shortens paths. This may sometimes be at the expense of cells, however, since some of the boxes shown in FIG. 3(*k*) may have to be replicated. The factoring to a fanin of 8, also produces shorter paths, but may increase the cell count, e.g., in a dual-rail technology in which a 4-way NOR/OR required one cell and an 8-way required two cells. This will be explained in more detail with reference to FIGS. 8(*a*) and 8(*b*).

In a particular technology, there may be a number of different primitives or "books" having different fan-in capabilities, and different books may include different numbers of cells. For example, an 8-way NAND gate may use two cells while a 4-way NAND gate may use one cell. If 8-way NAND gates are used, e.g., to combine ten different inputs in two combinations with four inputs common to each combination, the result may be as shown in FIG. 8(*a*). Each book would receive seven inputs, and a total of four cells would be used.

If fan-in is limited to a value of 4, the same logic could be implemented as shown in FIG. 8(*b*) using three 4-way books. Although the number of books has increased, each book includes only one cell, so that the cell count decreases from four to three. However, the

cell count decrease is at the expense of increasing the logic depth by one level.

In the "normal" or "small" scenarios, it is worthwhile to set the fan-in value to 4 and to factor in an attempt to take advantage of the cell reduction which may be realized by using the smaller books. In the "fast" scenario, however, the increase in logic depth accompanying the use of the smaller books is unacceptable, and the fanin is instead set to the maximum allowable fan-in after the NOCHANGE loop.

Thus, the simplification program for the "fast" scenario can be functionally described as follows:

LOGIC DEPTH REDUCTION LOOP
A1. REDUCE LOGIC DEPTH
A2. REMOVE REDUNDANCY
A3. ELIMINATE COMMON TERMS
A4. REDUCE LOGIC DEPTH WHILE INCREASING CELL COUNT
B. INTRODUCE DOT PATTERNS AND FACTOR TO REDUCE FANIN TO MAXIMUM ALLOWED BY TECHNOLOGY
C. REMOVE REDUNDANCY
D. ELIMINATE COMMON TERMS
E. ADJUST LEVEL TO MAXIMUM PERMITTED FANIN

The search strategy for the above transformations is to scarch the interconnected boxes of the date-base in sequence, looking for a pattern to which the transform may apply. The search is done for each transform in an efficient way, e.g., NTR2 searches the entire logic design for a one-input inverter, since this is faster than examining each multi-way NAND or NOR to determine if an inverter precedes or follows it.

After the simplification sequence described above, transformations are applied to the logic to map the NAND or NOR implementation to the target technology, simplify the technology-specific implementation, and enforoe technology-specific restrictions. This is performed at level 108 in FIG. 2. The transformations applied at level 108 may be generally described as follows, although their exact implementation will depend on the target technology

Technology-specific transforms may preferably be applied in the following order: first, generic NAND/-NOR gates are mapped to their counterparts in the target technology. If the fan-in of a gate is too high and there is no corresponding primitive in the technology, a tree of primitives must be built to produce the same logical function. REG's, the generic latches, are mapped to the technology-specific latches. In general, the technology-specific latches have a limited number of pins for data values. If more data values are gated into the latch than can be accomodated, extra "ports" must be connected to the latch in a manner prescribed for the technology. SENDER's and RECEIVER's are mapped to their technology-specific counter parts.

Second, if the target technology is dual-rail, dual-rail books are introduced. With both positive and negative phases available from each gate, all inverters (except those on chip inputs) are removed and their output signals connected to the opposite phase of the source of their input signals. Third, technology-specific "tricks" are introduced, e.g., special books, drivers, receivers, etc., which were not known at the time of the generic transformation. These implement certain functions, such as XOR, combinations of driver and logic functions, combinations of receiver and logic funtions, combinations of latch and receiver, etc., using fewer cells than the primitive NAND or NOR implementation. The pattern of technology-specific NAND's or NOR's is searched for and replaced by the appropriate block. In FIG. 7(*a*), three cells can be replaced with a single NAND in the target technology having a built-in receiver.

If dots, i.e., wired AND's or OR's, are allowed in the target technology, patterns implementing +AND or +OR functions are located. If the inputs of these patterns have no fan-out, the pattern is replaced by a dot, e.g., as shown in FIG. 7(*b*). Dots can also be introduced to reduce fan-in as shown in FIG. 7(*c*). After dots are introduced, more special books may be present and are searched for again.

Next, fan-out is adjusted to meet constraints. Fanout limits are specified for technology-specific box types and output pins of those boxes. Fan-out is brought within these limits by replicating the violating box and distributing some of its fan-out to the copy (parallel repowering) or driving some of the fan-out with a repowering +OR or +AND function in front of the violating box (serial repowering). Additional dual-rail books are added after fan-out adjustment, but not so as to violate fan-out constraints.

Next, clock signals are introduced as chip inputs and distributed to the latches of the chip according to technology-specific requirements for clock distribution. Depending on the technology, this requires clock balancing and introduction of special clock drivers.

Next, path lengths back from latch-to-latch and from chip output to chip input are analyzed. Long paths are first shortened by rearranging fan-in and fan-out repowering, introducing dots (even at the cost of cell count), "undoing" factoring transformations performed at a higher level and introducing high-power books. Short paths are then padded to meet minimum path length requirements.

The fan-out adjustment is then repeated to correct fan-out violations which may have resulted from the path length correction.

Finally, scan-in and scan-out pins are introduced and the latches are linked together in an LSSD scan ring. A chip-in-place test and/or inhibit signals are introduced for chip outputs where required. Since this may introduce fan-out violations, fan-out adjustment is repeated.

An example of a hardware conversion and simplification program for a NOR technology following the above-described sequence may be as follows:

APPLY GENHW, CLEANUP;
APPLY DUAL (NO LIMIT), CLEANUP;
APPLY OPTDRIVE (SMALL OR FAST), CLEANUP,
OPTXOR (SMALL OR FAST), CLEANUP;
APPLY GENDOT;
APPLY OPTDRIVE (SMALL OR FAST), CLEANUP, OPTXOR (SMALL OR FAST), CLEANUP;
APPLY FANOUT, DUAL, CLEANUP;
APPLY CLOCK;
APPLY TIMINE;
APPLY FANOUT, DUAL, CLEANUP
APPLY SCANP, FANOUT;

GENHW maps generic gates to hardware primitives. Since fan-in has been adjusted at the end of the NAND/NOR simplification, much of this step is merely one-to-one mapping.

DUAL removes necessary inverters in dual-rail technologies by absorbing the inverters into other gates

which already have positive and negative phases available. This transform will normally be applied so as to exceed the fan-out limit. However, with the NOLIMIT option this transform will always apply.

OPTDRIVE takes advantage of a technology-specific book available, i.e., a driver book with built-in NOR capability. As shown in FIG. 9(*a*), the logic design may at this point include a NOR gate with a branched output with one branch going to a driver. Since both functions can be served by a single book in the target technology, the arrangement of FIG. 9(*b*) can be substituted. However, while this may be desirable in "normal" and "small" scenarios, there is a sacrifice in speed. Thus, for a "fast" scenario, the transformation is to the arrangement shown in FIG. 9(*c*) which provides for "parallel" operation and therefore higher speed at the expense of cell count.

OPTXOR takes advantage of a further technologys-pecific book, i.e., the XOR book. This transformation searches for a pattern of NOR gates providing the XOR function, e.g., as shown in FIG. 10(*a*), and substitutes the XOR book as shown in FIG. 10(*b*). Again, however, the transformation to FIG. 10(*c*) is employed in the "fast" scenario.

GENDOT introduces dotting in such a manner as to both eliminate gates and reduce fan-in. E.g., the transformation shown in FIG. 7(*b*) will eliminate gates 15 and 16 while the transformation shown in FIG. 7(*c*) will not eliminate gate 17 but will reduce the fan-in to that gate. This may save cells by permitting the use of a smaller book in the target technology and by allowing other transforms to apply. Since GENDOT changes the logic, OPTDRIVE and OPTXOR are applied again to search for more special books which may now exist.

FANOUT is applied to reduce the fan-out to the allowed limit. Note that the first half of the above hardware level simplification program is run without regard to fan-out limitations, as even the DUAL transform is applied with its NOLIMIT option. The various transformations may have caused fan-out violations which should be corrected by applying FANOUT in the manner discussed above. DUAL is then applied again, but this time so as not to violate fan-out constraints.

CLOCK is applied to distribute clock signals according to technology specific requirements in a manner known in the art.

TIMING is applied to correct path lengths by rearranging fan-in and fan-out trees, introducing more dots and chaning power levels to shorten long path lengths, and inserting pad logic to lengthen the short paths, as necessary. After TIMING is applied, fan-out adjustment is again performed to correct any violations which may have resulted from the timing correction, and DUAL is again run, within fan-out constraints, to take advantage of changes made during fan-out adjustment.

Finally, SCANP is applied to link the registers in a LSSD scan path. Fan-out is again adjusted to correct any violations which may have resulted from SCANP.

The logic synthesis system of this invention employs three different levels of simplification between the original specification and the final implementation: high level simplifications, NAND/NOR level simplifications and technology specific simplifications. Several of the transforms at the three different levels are analogous, differing only in the types of boxes to which they apply, so that simplifications not made at one level would be caught later. This may appear redundant, but the application of transforms as early as possible reduces the size of the implementation and helps prevent a greater explosion in size when, e.g., conversion to NANDs takes place.

A significant advantage of the present invention resides in its adaptibility to more than one technology, requiring modifications to only a part of the system and leaving the technology-independent portions intact. This makes the synthesis process according to the present invention useful in synthesizing logic in a number of different technologies, and in fact facilitates the remapping from one technology to another in an efficient manner. Rather than merely mapping hardware primitives one-to-one from one technology to another, a first technology implementation is abstracted to a technology-independent level, e.g., from a TTL chip implementation to a NAND level implementation with generic registers, drivers and receivers. The NAND implementation can be mapped to a NOR level implementation in a straight-forward manner, with the NOR level simplification being performed in the manner described above with reference to level **106** in FIG. 2. The hardware mapping and simplification can then be performed in the manner described with reference to level **108** in FIG. 2. This enables the remapping to take advantage of simplifications which may be available at the NOR level.

Some of the work described in the earlier-cited publications concerned a synthesis process beginning with a behavioral description and producing technology-independent implementations of boolean equations. These processes did not take advantage of the target technology. Other work has centered on the synthesis of the data-flow portion of a machine, synthesis from a high-level behavioral description to a register-transfer description, and implementation of control logic in microcode or programmable logic arrays. In contrast, the present invention provides the following significant features:

First, the present invention uses local transformations at several levels of description, passing through technologyindependent levels of description to a technology-specific description. This enhances the simplification while also facilitating the re-implementation of a design in a different technology.

Second, the specific sequences of simplifying transformations and the conditions associated with them have been found to provide acceptable results in normal, fast and small scenarios, thus making automated logic synthesis practical.

Further, timing, driver and other interface constraints are used at the hardware level to generate logic meeting these requirements.

Still further, the automated logic synthesis process according to the present invention greatly facilitates timing analysis and correction of the design to remove path length problems.

What is claimed is:

1. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating, via said computer, a first logic circuit design in a first logic system in accordance with said description;

simplifying said first logic design via said computer;

converting, via said computer, said simplified first logic design to a second logic design in a second logic system requiring fewer different logic operators than said first logic system, said second logic system comprising a plurality of interconnected cells and performing equivalent functions;

simplifying, via said computer, said second logic design, said step of simplifying said second logic design comprising the steps of: applying a depth reduction sequence of logic transformations for reducing the depth of said second logic design; and subsequently applying a size reduction sequence of logic transformations for reducing the size while possibly increasing the depth of said second logic design; and

converting, via said computer, said simplified second logic design to said desired technology.

2. A method as defined in claim 1, wherein said step of applying said depth reduction sequence of logic transformations comprises: applying a first logic transformation set for reducing logic depth; applying a second logic transformation set for reducing redundancy; and applying a third logic transformation set for eliminating common terms.

3. A method as defined in claim 2, wherein said step of applying said size reduction sequence of logic transformations comprises: applying a fourth logic transformation set for reducing logic size while possibly increasing logic depth; applying a fifth logic transformation set for reducing redundancy; and applying a sixth logic transformation set for eliminating common terms.

4. A method as defined in claim 3, wherein said second and fifth logic transformation sets include at least one common logic transformation. said common logic transformation being applied in said depth reduction sequence regardless of whether said common logic transformation will increase the number of cells in said second logic design and being applied in said size reduction sequence only if it will not result in an increase in said number of cells.

5. A method as defined in claim 2, wherein said step of applying said depth reduction sequence of logic transformations further comprises applying a fourth logic transformation set (e.g.. NTR8), following said third logic transformation set, for further reducing said logic depth while increasing the number of cells in said second logic design.

6. A method as defined in claim 3, wherein said converting step comprises converting said simplified second logic design to a hardware logic design in said desired technology represented as a plurality of hardware primitives. said method further comprising the step of simplifying said hardware design, said hardware simplifying step comprising:

applying a first hardware transformation set for substituting technology-specific books for predetermined patterns of said hardware primitive;

dotting signal lines to decrease the number of components in said hardware logic design, and to decrease fan-in in some portions of said hardware logic design even if the number of components in said portions is not decreased;

applying said first hardware transformation set;

correcting fan-out in said hardware logic design to a desired value;

adjusting path lengths in said hardware logic design; and

correcting fan-out to said desired value.

7. A method as defined in claim 1, wherein said step of applying said size reduction sequence of logic transformations comprises applying a first logic transformation (e.g., FACTORN) for reducing a fan-in characteristic of some portions of said second logic design in accordance with a first fan-in value.

8. A method as defined in claim 1, further comprising the step of applying a cell reduction sequence of logic transformations for reducing the number of cells in said second logic design, said cell reduction sequence being applied between said depth reduction and size reduction sequences.

9. A method as defined in claim 8, wherein said cell reduction sequence of logic transformations includes a first set of logic transformations followed by a second set of logic transformations, said second set of logic transformations comprising said depth reduction sequence of logic transformations.

10. A method as defined in claim 8, wherein said desired technology has a maximum allowable fan-in value, said step of applying said size reduction sequence of logic transformations comprising applying a first logic transformation (e.g., FACTORN) for reducing a fan-in characteristic of some portions of said second logic design in accordance with a desired fan-in value less than said maximum allowable fan-in value.

11. A method as defined in claim 10, further comprising the step of correcting the fan-in characteristics of said second logic design in accordance with said maximum allowable fan-in value subsequent to application of said size reduction sequence of logic transformations.

12. A method as defined in claim 1, wherein said depth reduction sequence of logic transformations is applied a plurality of times prior to applying said size reduction sequence of logic transformations.

13. A method as defined in claim 1, wherein said first logic design is implemented in AND/OR logic and said second logic design is implemented in NAND logic.

14. A method as defined in claim 1, wherein said first logic design is implemented in AND/OR logic and said second logic design is implemented in NOR logic.

15. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating, via said computer, a first logic circuit design in a first logic system in accordance with said description;

simplifying said first logic design via said computer;

converting, via said computer, said first logic design to a second logic design in a second logic system requiring fewer different logic operators than in said first logic design, said second logic design comprising a plurality of interconnected cells and performing equivalent functions as said first logic design;

simplifying said second logic design via said computer;

converting, via said computer, said simplified second logic design to a hardware design in said desired technology comprising a plurality of interconnected hardware components; and

simplifying said hardware design via said computer, said step of simplifying said hardware design comprising: applying a first hardware transformation

set for substituting technology specific components for predetermined patterns of said hardware;

dotting signal lines via said computer, to decrease the number of components in said hardware logic design and to decrease fan-in in some portions of said hardware logic design even if the number of components is said portions is not decreased;

adjusting path lengths in said hardware logic design via said computer; and

correcting fan-out in said hardware logic design to a desired value via said computer.

16. A method as defined in claim 15, wherein said step of simplifying said hardware design further comprises applying said first hardware transformation set again after said dotting step but before said adjusting step.

17. A method as defined in claim 16, wherein said step of simplifying said hardware logic design further comprises the step of correcting said fan-out in said hardware logic design after said second application of said first hardware transformation set and before said adjusting step

18. A method as defined in claim 15, wherein said hardware logic design includes inverters receiving and inverting outputs from associated components, and wherein, when said desired technology is a dual-rail technology, said step of simplifying said hardware design further comprises the step of applying a dual-rail transformation for removing some of said inverters by substituting for said inverter and opposite-phase output signal available from its associated component, said dual-rail conversion transformation being applied both prior to said step of applying said first hardware transformation and subsequent to said step of correcting fan-out.

19. A method as defined in claim 18, wherein said dual-rail conversion transformation is applied prior to said step of applying said first hardware transformation without regard to the effect of said dual-rail conversion transformation on fan-out characteristics of said hardware logic design, said dual-rail conversion transformation being applied after said step of correcting fan-out only to the extent that application of said dualrail conversion transformation will not result in fan-out exceeding said desired value.

20. A method as defined in claim 15, wherein said hardware logic design includes inverters receiving and inverting outputs from associated components, and wherein, when said desired technology is a dual-rail technology, said step of simplifying said hardware design further comprises the step of applying a dual-rail comversion transformation, prior to said step of applying said first hardware transformation, for removing some of said inverters by substituting for said some inverters an oppositephase output available from their associated components.

21. A method as defined in claim 20, wherein said step of simplifying said hardware design further comprises the step of applying said dual-rail conversion transformation subsequent to said step of correcting fan-out.

22. A method as defined in claim 21, wherein said dual-rail conversion transformation is applied prior to said application of said first hardware transformation without regard to the effect of said dual-rail conversion transformation on fan-out characteristics of said hardware logic design, and is applied subsequent to said step of correcting fan-out only to the extent that application of said dual-rail conversion transformation will not result in fan-out exceeding said desired value.

23. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating a first logic circuit design via said computer, in accordance with said description;

simplifying said first logic design via said computer;

converting, via said computer, said first logic design to a second logic design in a second logic system requiring fewer different logic operators than in said first logic design;

simplifying said second logic design via said computer;

converting, via said computer, said simplified second logic design to a hardware design in said desired technology comprising a plurality of interconnected hardware components and including inverters for receiving and inverting output signals from associated ones of said components; and

simplifying said hardware design via said computer, said step of simplifying said hardware logic design comprising: applying a dual-rail conversion transformation for removing some of said inverters by subetituting for said some inverters an opposite phase output signal available from their associated components, said dual-rail conversion transformation being applied without regard to the effect on fan-out characteristics of said hardware logic design; applying a first hardware transformation set for substituting technology-specific components; dotting signal lines in said hardware logic design; adjusting path lengths in said hardware logic design; correcting fan-out in said hardware logic design to a desired value; and applying said dual-rail conversion transformation only to the extent that it does not result in a fan-out exceeding said desired value.

24. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating, via said computer, a first logic circuit design accordance with said description;

simplifying said first logic design via said computer;

converting, via said computer, said first logic design to a second logic design in a second logic system requiring fewer different logic operators than in said first logic design;

simplifying said second logic design via said computer;

converting, via said computer, said simplified second logic design to a hardware design in said desired technology comprising a plurality of interconnected hardware components; and

simplifying said hardware design via said computer, said step of simplifying said hardware logic design compriaing selectively applying first first or second hardware transformation sets for substituting technology-specific components for predetermined patterns of said hardware components, said first hardware transformation set resulting in fewer components than said second hardware transformation set and said second hardware transformation set resulting in higher-speed logic than said first hardware transformation set.

* * * * *

EXHIBIT 7

matching a protection key associated with a store reference to main storage with a storage key associated with each block of main storage. See also fetch protection.

store support procedure A procedure that assists personnel in administrative, operational, and managerial operations apart from customer checkout.

store through cache In a processing unit, a store (write) operation, in which data are immediately put into both cache and main storage locations.

storing (1) The action of placing data into a storage device. (2) To place data into a storage device. (3) To retain data in a storage device. (T)

storyboard In multimedia applications, a visual representation of the script, showing a picture of each scene and describing its corresponding audio. Synonymous with slide show presentation.

storyboarding In multimedia applications, producing a sequence of still images, such as titles, graphics, and images, to work out the visual details of a script.

STP Stop character.

STR Synchronous transmitter receiver.

straight line coding (1) A set of instructions without loops. (I) (A) (2) Programming technique in which loops are avoided by unwinding. (I) (A)

stratified language (1) A language that cannot be used as its own metalanguage; for example, FORTRAN. (I) (A) (2) Contrast with unstratified language.

streak A narrow area on a printed sheet that is either darker or lighter than desired. Contrast with gray bar, spot.

stream (1) To send data from one device to another. (2) See data stream.

stream data transmission In PL/I, the transmission of data in which the organization of the data into records is ignored and the data is treated as though it were a continuous stream of individual data values in character form. Contrast with record data transmission.

stream editor In text processing, a text editor that treats the entire text as a single string, even when the string is broken into lines for viewing purposes. (T) (A)

streamer Synonym for streaming tape drive.

stream file In BASIC, a file on disk in which data is read and written in consecutive fields without record boundaries. Contrast with record file.

streaming (1) A condition in which a device remains in a transmit state for an abnormal length of time. (2) A method of writing and reading data on magnetic tape as continuous fields without record boundaries.

streaming tape drive A magnetic tape unit especially designed to make a nonstop dump or restore of magnetic disks without stopping at interblock gaps. Synonymous with streamer. (T) Contrast with start-stop tape drive.

streaming tape recording A method of recording on magnetic tape that maintains continuous tape motion without the requirement to start and stop within the interrecord gap. (A)

stream mode A method of sending and receiving data in which records are defined as a stream of data without boundaries.

strength member In an optical cable, material that can be located either centrally or peripherally and that functions as a strain relief.

stress patterns In printing, severe print-quality standard patterns used to test print quality.

strict type checking In C language, checking data types for compliance with the rules of C language more strictly than C compiler checking.

strike In videotaping, to clear away, remove, or dismantle anything on the set.

strikeover A character entered in a space currently occupied by another character.

string (1) A sequence of elements of the same nature, such as characters considered as a whole. (T) (2) In programming languages, the form of data used for storing and manipulating text. (3) In XL Pascal, an object of the predefined type STRING. (4) In the AS/400 system, a group of auxiliary storage devices connected in a series on the system. The order and location in which each device is connected to the system determines the physical address of the device. (5) In PL/I, a sequence of characters or bits that is treated as a single data item. (6) In SQL, a character string. (7) See alphabetic string, binary element string, bit string, character string, compound string, conformant string, literal string, mixed string, null string, pattern string, symbol string, text string, unit string.

string constant In Pascal, a string whose value is fixed by the compiler.

EXHIBIT 8

Ninth New Collegiate Dictionary

A Merriam-Webster®

MERRIAM-WEBSTER INC., *Publishers*
Springfield, Massachusetts, U.S.A.

RCL011411

A GENUINE MERRIAM-WEBSTER

The name *Webster* alone is no guarantee of excellence. It is used by a number of publishers and may serve mainly to mislead an unwary buyer.

A Merriam-Webster® is the registered trademark you should look for when you consider the purchase of dictionaries or other fine reference books. It carries the reputation of a company that has been publishing since 1831 and is your assurance of quality and authority.

Copyright © 1987 by Merriam-Webster Inc.

Philippines Copyright 1987 by Merriam-Webster Inc.

Library of Congress Cataloging in Publication Data
Main entry under title:

Webster's ninth new collegiate dictionary.

Based on Webster's third new international dictionary.
Includes index.
1. English language—Dictionaries. I. Merriam-Webster Inc.
PE1628.W5638 1987 423 86-23801
ISBN 0-87779-508-8
ISBN 0-87779-509-6 (indexed)
ISBN 0-87779-510-X (deluxe)

Webster's Ninth New Collegiate Dictionary principal copyright 1983

COLLEGIATE trademark Reg. U.S. Pat. Off.

All rights reserved. No part of this book covered by the copyrights hereon may be reproduced or copied in any form or by any means—graphic, electronic, or mechanical, including photocopying, taping, or information storage and retrieval systems—without written permission of the publisher.

Made in the United States of America

2526RMcN87

RCL011412

completely closed 10 : a depression in the face of an animal at the junction of forehead and muzzle — see DOG illustration
²stop *adj* (1594) : serving to stop : designed to stop ⟨~ line⟩ ⟨~ signal⟩
stop-and-go \ˌstäp-ən-'gō, -ᵊm-, *attributively* -ˌgō\ *adj* (1925) : of, relating to, or involving frequent stops; *esp* : controlled or regulated by traffic lights ⟨~ driving⟩
stop bath *n* (ca. 1918) : an acid bath used to check photographic development of a negative or print
stop-cock \'stäp-ˌkäk\ *n* (1584) : a cock for stopping or regulating flow (as through a pipe)
stop down *vt* (ca. 1891) : to reduce the effective aperture of (a lens) by means of a diaphragm
¹stope \'stōp\ *n* [prob. fr. LG *stope*, lit., step; akin to OE *stæpe* step — more at STEP] (1747) : a usu. steplike excavation underground for the removal of ore that is formed as the ore is mined in successive layers
²stope *vb* **stoped; stop·ing** *vt* (1778) : to mine by means of a stope ~ *vt* : to extract (ore) from a stope — **stop·er** *n*
stop-gap \'stäp-ˌgap\ *n* (1684) : something that serves as a temporary expedient : MAKESHIFT *syn* see RESOURCE
stop knob *n* (1887) : one of the handles by which an organist draws or shuts off a particular stop
stop-light \'stäp-ˌlīt\ *n* (1926) 1 : a light on the rear of a motor vehicle that is illuminated when the driver presses the brake pedal 2 : TRAFFIC SIGNAL
stop order *n* (ca. 1891) : an order to a broker to buy or sell respectively at the market when the price of a security advances or declines to a designated level
stop out \(ᵊ)stäp-'aůt\ *vi* [¹*stop* + *out* (as in *drop out*)] (1973) : to withdraw temporarily from enrollment at a college or university — **stop-out** \'stäp-ˌaůt\ *n*
stop-over \'stäp-ˌō-vər\ *n* (1885) 1 : a stop at an intermediate point in one's journey 2 : a stopping place on a journey
stop-page \'stäp-ij\ *n* (15c) : the act of stopping : the state of being stopped : HALT, OBSTRUCTION
stop payment *n* (ca. 1919) : a depositor's order to a bank to refuse to honor a specified check drawn by him
¹stop-per \'stäp-ər\ *n* (15c) 1 : one that brings to a halt or causes to stop operating or functioning : CHECK; as a : a playing card that will stop the running of a suit b : a baseball pitcher depended on to win important games or to stop a losing streak; *also* : an effective relief pitcher 2 : one that closes, shuts, or fills up; *specif* : something (as a bung or cork) used to plug an opening
²stopper *vt* **stop·pered; stop·per·ing** \-(ə-)riŋ\ (ca. 1769) : to close or secure with or as if with a stopper
stopper knot *n* (1860) : a knot used to prevent a rope from passing through a hole or opening
¹stop·ple \'stäp-əl\ *n* [ME *stoppell*, fr. *stoppen* to stop] (14c) : something that closes an aperture : STOPPER, PLUG
²stopple *vt* **stop·pled; stop·pling** \-(ə-)liŋ\ (1795) : STOPPER
stop street *n* (ca. 1930) : a street on which a vehicle must stop just before entering a through street
stop-watch \'stäp-ˌwäch\ *n* (1737) : a watch having a component (as a hand) that can be started and stopped at will for exact timing (as of a race)
stor·age \'stōr-ij, 'stór-\ *n* (1612) 1 a : space or a place for storing b : an amount stored c : MEMORY 4 2 a : the act of storing : the state of being stored; *esp* : the safekeeping of goods in a depository (as a warehouse) b : the price charged for keeping goods in a storehouse 3 : the production by means of electric energy of chemical reactions that when allowed to reverse themselves generate electricity again without serious loss
storage cell *n* (1881) : a cell or connected group of cells that converts chemical energy into electrical energy by reversible chemical reactions and that may be recharged by passing a current through it in the direction opposite to that of its discharge — called also *storage battery*
sto·rax \'stō(ə)r-ˌaks, 'stó(ə)r-\ *n* [ME, fr. LL, alter. of L *styrax*, fr. Gk] (14c) 1 a : a fragrant balsam obtained from the bark of an Asian tree (*Liquidambar orientalis*) of the witch-hazel family that is used as an expectorant and sometimes in perfumery — called also *Levant storax* b : a balsam from the sweet gum that is similar to storax 2 : any of a genus (*Styrax* of the family Styracaceae, the storax family) of trees or shrubs with usu. hairy leaves and flowers in drooping racemes — compare BENZOIN
¹store \'stō(ə)r, 'stó(ə)r\ *vt* **stored; stor·ing** [ME *storen*, fr. OF *estorer* to construct, restore, store, fr. L *instaurare* to renew, restore, fr. *in-* + *-staurare* (akin to Gk *stauros* stake) — more at STEER] (13c) 1 : FURNISH, SUPPLY; *esp* : to stock against a future time ⟨~ a ship with provisions⟩ 2 : LAY AWAY, ACCUMULATE ⟨~ vegetables for winter use⟩ ⟨an organism that absorbs and ~s DDT⟩ 3 : to place or leave in a location (as a warehouse, library, or computer memory) for preservation or later use or disposal 4 : to provide storage room for : HOLD ⟨elevators for *storing* surplus wheat⟩ — **stor·able** \'stōr-ə-bəl, 'stór-\ *adj*
²store *n* (13c) 1 a : something that is stored or kept for future use b *pl* : articles (as of food) accumulated for some specific object and drawn upon as needed : STOCK, SUPPLIES c : something that is accumulated d : a source from which things may be drawn as needed : a reserve fund 2 : STORAGE — usu. used with *in* ⟨when placing eggs in by a partner's opinion⟩ 4 : a large quantity, supply, or number : ABUNDANCE 5 a : STOREHOUSE, WAREHOUSE b *chiefly Brit* : MEMORY 4 6 : a business establishment where usu. diversified goods are kept for retail sale ⟨grocery ~⟩ — compare SHOP — **in store** : in a state of imminence
³store *adj* (1602) 1 *or* **stores** : of, relating to, kept in, or used for a store 2 : purchased from a store as opposed to being natural or homemade : MANUFACTURED, READY-MADE ⟨~ clothes⟩ ⟨~ bread⟩
store-bought \'stō(ə)r-ˌbȯt, 'stó(ə)r-\ *adj* (1905) : STORE 2
store cheese *n* [fr. its being a staple article stocked in grocery stores] (1863) : CHEDDAR
¹store·front \'stō(ə)r-ˌfrənt, 'stó(ə)r-\ *adj* (1937) 1 : of, relating to, or characteristic of a storefront church ⟨a ~ evangelist⟩ 2 : occupying a room or suite of rooms in a store building at street level and immediately behind a storefront ⟨a ~ school⟩ 3 : of, relating to, or being outreach professional services ⟨~ lawyers⟩ ⟨~ day-care center⟩ ⟨~ hospitals⟩
²storefront *n* (1943) 1 : the front side of a store or store building facing a street 2 : a building, room, or suite of rooms having a storefront
storefront church *n* (1937) : a city church that utilizes storefront quarters as a meeting place and that usu. holds services of a highly emotional nature
store·house \'stō(ə)r-ˌhaůs, 'stó(ə)r-\ *n* (14c) 1 : a building for storing goods (as provisions) : MAGAZINE, WAREHOUSE 2 : an abundant supply or source : REPOSITORY
store·keep·er \-ˌkē-pər\ *n* (1618) 1 : one that has charge of supplies (as military stores) 2 : one that operates a retail store
store·room \-ˌrüm, -ˌrům\ *n* (1746) 1 : a room or space for the storing of goods or supplies 2 : STOREHOUSE 2
store·ship \-ˌship\ *n* (ca. 1693) : a ship used to carry supplies
store-wide \-'wīd\ *adj* (ca. 1937) : including all or most merchandise in a store ⟨a ~ sale⟩
¹sto·ried \'stōr-ēd, 'stór-\ *adj* (15c) 1 : decorated with designs representing scenes from story or history ⟨a ~ frieze⟩ ⟨a ~ tapestry⟩ 2 : having an interesting history : celebrated in story or history
²storied *or* **sto·reyed** \'stōr-ēd, 'stór-\ *adj* (1624) : having stories ⟨a two-*storied* house⟩
stork \'stó(ə)rk\ *n* [ME, fr. OE *storc*; akin to OHG *storah* stork, OE *stearc* stiff — more at STARK] (bef. 12c) : any of various large mostly Old World wading birds (family Ciconiidae) that have long stout bills and are related to the ibises and herons
storks-bill \'stórks-ˌbil\ *n* (ca. 1562) : any of several plants of the geranium family with elongate beaked fruits: a : PELARGONIUM b : ALFILARIA; *also* : a related plant (genus *Erodium*)
¹storm \'stó(ə)rm\ *n, often attrib* [ME, fr. OE; akin to OHG *sturm* storm, OE *styrian* to stir] (bef. 12c) 1 a : a disturbance of the atmosphere marked by wind and usu. by rain, snow, hail, sleet, or thunder and lightning b : a heavy fall of rain, snow, or hail c (1) : wind having a speed of 64 to 72 miles (103 to 116 kilometers) per hour (2) : WHOLE GALE — see BEAUFORT SCALE table d : a serious disturbance of any element of nature 2 : a disturbed or agitated state : a sudden or violent commotion 3 : a heavy discharge of objects (as missiles) 4 : a tumultuous outburst 5 a : PAROXYSM, CRISIS b : a sudden heavy influx or onset 6 : a violent assault on a defended position — **by storm** : by or as if by employing a bold swift frontal movement esp. with the intent of defeating or winning over quickly
²storm *vi* (15c) 1 a : to blow with violence b : to rain, hail, snow, or sleet 2 : to attack by storm ⟨~*ed* ashore at zero hour⟩ 3 : to be in or to exhibit a violent passion : RAGE ⟨~*ing* at the unusual delay⟩ 4 : to rush about or move impetuously, violently, or angrily ⟨the mob ~*ed* through the streets⟩ ~ *vt* : to attack, take, or win over by storm ⟨~ a fort⟩ *syn* see ATTACK
storm and stress *n, often cap both Ss* (1855) : STURM UND DRANG
storm boat *n* (1942) : a light fast craft used to transport attacking troops across streams
storm-bound \'stó(ə)rm-'baůnd\ *adj* (1830) : cut off from outside communication by a storm or its effects : stopped or delayed by storms
storm cellar *n* (ca. 1902) : CYCLONE CELLAR
storm door *n* (1878) : an additional door placed outside an ordinary outside door for protection against severe weather
storm petrel *n* (ca. 1833) : any of various small petrels; *esp* : a small sooty black white-marked petrel (*Hydrobates pelagicus*) frequenting the north Atlantic and Mediterranean

storm petrel

storm trooper *n* (1935) 1 : a member of a private Nazi army notorious for aggressiveness, violence, and brutality 2 : one that resembles a Nazi storm trooper
storm window *n* (ca. 1888) : a sash placed outside an ordinary window as a protection against severe weather — called also *storm sash*
stormy \'stōr-mē\ *adj* **storm·i·er; -est** (13c) 1 : relating to, characterized by, or indicative of a storm ⟨a ~ day⟩ ⟨a ~ autumn⟩ 2 : marked by turmoil or fury ⟨a ~ life⟩ ⟨a ~ conference⟩ — **storm·i·ly** \'stōr-mə-lē\ *adv* — **storm·i·ness** \-mē-nəs\ *n*
stormy petrel *n* (ca. 1776) 1 : STORM PETREL 2 a : one fond of strife b : a harbinger of trouble
¹sto·ry \'stōr-ē, 'stór-\ *n, pl* **stories** [ME *storie*, fr. OF *estorie*, fr. L *historia* — more at HISTORY] (13c) 1 *archaic* : HISTORY 1,3 2 a : an account of incidents or events b : a statement regarding the facts pertinent to a situation in question c : ANECDOTE; *esp* : an amusing one 3 a : a fictional narrative shorter than a novel; *specif* : SHORT STORY b : the intrigue or plot of a narrative or dramatic work 4 : a widely circulated rumor 5 : LIE, FALSEHOOD 6 : LEGEND, ROMANCE 7 : a news article or broadcast
²story *vt* **sto·ried; sto·ry·ing** (15c) 1 *archaic* : to narrate or describe in story 2 : to adorn with a story or a scene from history
³story *also* **sto·rey** \'stōr-ē, 'stór-ē\ *n, pl* **stories** *also* **storeys** [ME *storie*, fr. ML *historia* picture, story of a building, fr. L history, tale; prob. fr. pictures adorning the windows of medieval buildings] (15c) 1 a : the space in a building between two adjacent floor levels or between a floor and the roof b : a set of rooms in such a space c : a unit of measure equal to the height of the story of a building ⟨one ~ high⟩ 2 : a horizontal division of a building's exterior not necessarily corresponding exactly with the stories within
story·board \-ˌbō(ə)rd, -ˌbȯ(ə)rd\ *n* (ca. 1946) : a panel or series of panels on which is tacked a set of small rough drawings depicting consecutively the important changes of scene and action in a planned film or television show or act
¹sto·ry-book \-ˌbůk\ *n* (1711) : a book of stories ⟨~s for children⟩
²storybook *adj* (1910) : FAIRY-TALE
story line *n* (1946) : the plot of a story or play